The Smart Pet Feeder: Progress Report 2

A Progress Report on the Design and Building of an Automated Pet Feeder Capable of Preventing One Pet From Eating Another Pet's Food

Submitted to Professor Salah Badjou

on March 31, 2008

by

Rachel Heil	Kristine McCarthy	Filip Rege	Alexis Rodriguez-Carlson
802-338-0165	508-280-2562	617-230-0196	617-359-9019
heilr@wit.edu	mccarthyk8@wit.edu	regef@wit.edu	rodriguezcarls@wit.edu
68 Louis Prang St.	610 Huntington Ave.	8 Barton St.	143 Watertown St. #2
Boston, MA 02115	Box 1079	Somerville, MA 02144	Watertown, MA 02472
	Boston, MA 02115		

WENTWORTH INSTITUTE OF TECHNOLOGY

ELMC 461-03/04 ELECTROMECHANICAL DESIGN

Table of Contents:

Introduction:

(Project manager: Alexis Rodriguez-Carlson)

This is the second progress report* detailing the process of designing and building a prototype of the Smart Pet Feeder, an automated pet feeder that will be suitable for use by cats and small dogs. The Smart Pet Feeder will solve two problems that pet owners face. These problems are:

1. Making sure that each pet has access to a healthy amount of food throughout the day, regardless of the owner's schedule
2. Making sure that each pet eats only its own food

Though there are a variety of products on the market that solve the first problem, there are none that address the second. The Smart Pet Feeder will give pet owners a solution to both problems, thereby improving the lives of both pets and owners by allowing the owner to:

1. Reliably provide food to a pet at the time the owner wishes
2. Keep the pet from reaching the food stored for later feedings
3. Restrict an unauthorized pet (called the "forbidden pet") from accessing the food in the feeder

The Smart Pet Feeder will look like the model in Figure 1. It will consist of a tray that holds 6 cups of food and will be mounted to a motor. The motor and the base will be inside an enclosure that will display only one bowl of food at a time. At predetermined times (which are programmable by the owner) the tray will rotate and reveal a fresh cup of food.

* We have attempted to repeat as little information from the first progress report as possible, however some information is repeated for the sake of clarity.

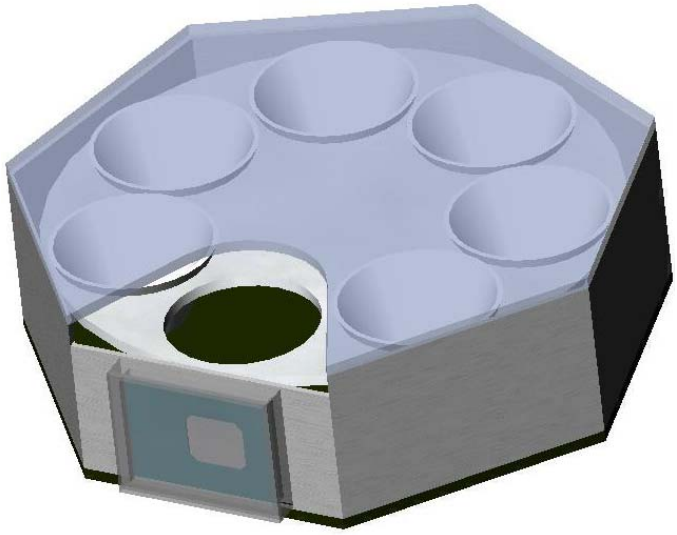


Figure 1: Model of the Smart Pet Feeder

The Smart Pet Feeder is designed to feed only one pet. To make sure that no forbidden pet eats the food in this feeder, there will be a radio frequency identification (RFID) reader mounted to the enclosure of the feeder in front of the revealed bowl (see Figure 1). This reader will be paired with a tag on the forbidden pet's collar. When the reader receives the signal from the tag it will trigger the motor to rotate so that the spot on the tray with no bowl, the "blank spot", is exposed, thus keeping the forbidden pet from eating.

The Smart Pet Feeder consists of three main systems: the Control System, the Motor System, and the Feeder Enclosure (see Figure 2). Each system has roles that it must fulfill. The Control System's roles are:

1. To rotate the tray to a new dish at a specified time
2. To rotate the tray to the blank spot if the forbidden pet approaches the feeder
3. To keep and display an accurate real-time clock
4. To allow the owner to easily set both the current time and the time the feeder will rotate to reveal fresh food

The Motor System's roles are:

1. To receive commands from the microcontroller
2. To be able to rotate the fully loaded tray an exact distance
3. To not allow the full weight of the tray to sit on the shaft of the motor

The Feeder Enclosure’s roles are to:

1. Be heavy enough so that a pet cannot turn it over
2. To close securely enough that the pet cannot open it and access the food
3. To be opened easily so that the owner can refill the dishes
4. To protect the electronics inside of it
5. To assure that all of the parts of the feeder that touch food are washable.

A block diagram showing all of the systems and how they interact with one another can be found in Figure 2.

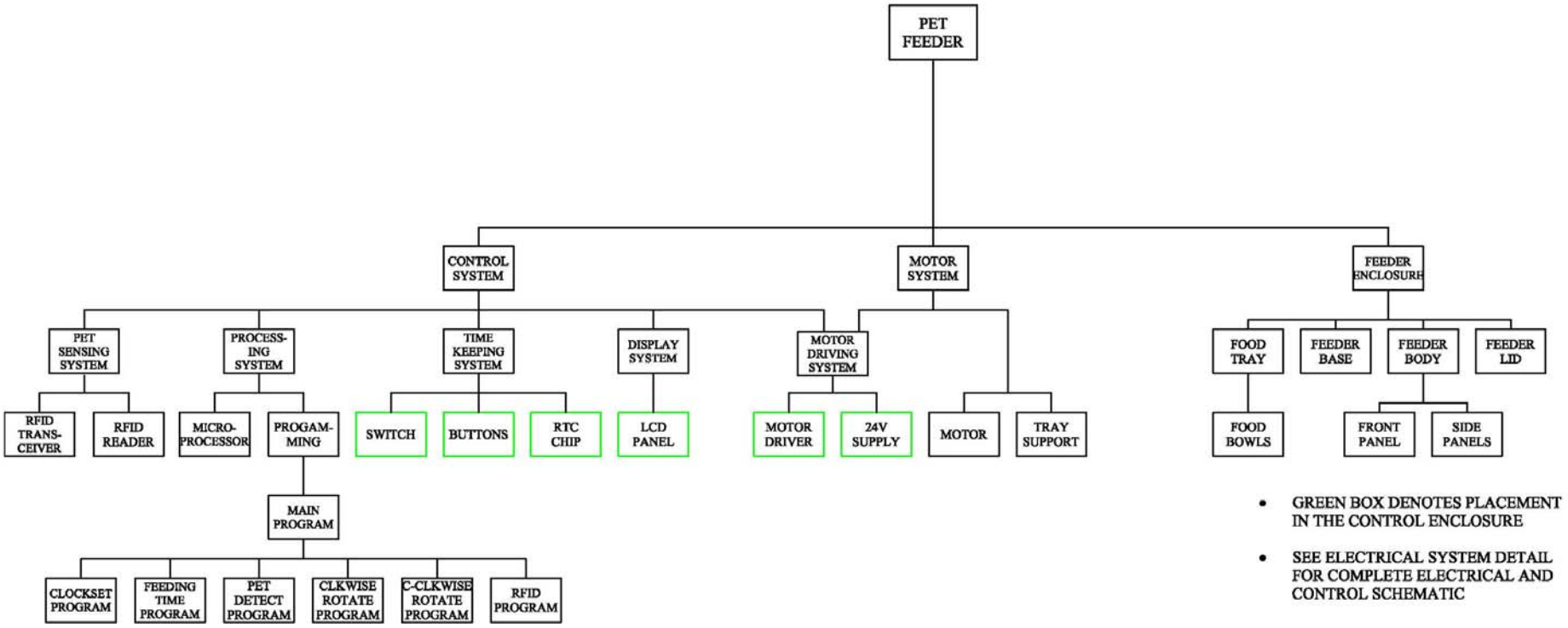


Figure 2: System Block Diagram

Progress-to-date:

Since the systems that make up the Smart Pet Feeder are so distinct, it is easiest to discuss them one at a time. This section will briefly recap the requirements for each section [1] and give an update on the progress of that system since the submittal of the first progress report [1]. It will start with the Control System, progress to the Motor System, and end with the Feeder Enclosure.

The Control System:

The Control System consists of several subsystems: the Pet Sensing System, the Processing System, the Time Keeping System, the Display System, and the Motor Driving System (this subsystem is also part of the Motor System and will be discussed later). This section will detail the requirements for each subsystem in Figure 3, the criteria used to select components, and discuss any changes in the status of this system since the progress report submitted on March 5, 2008 [1].

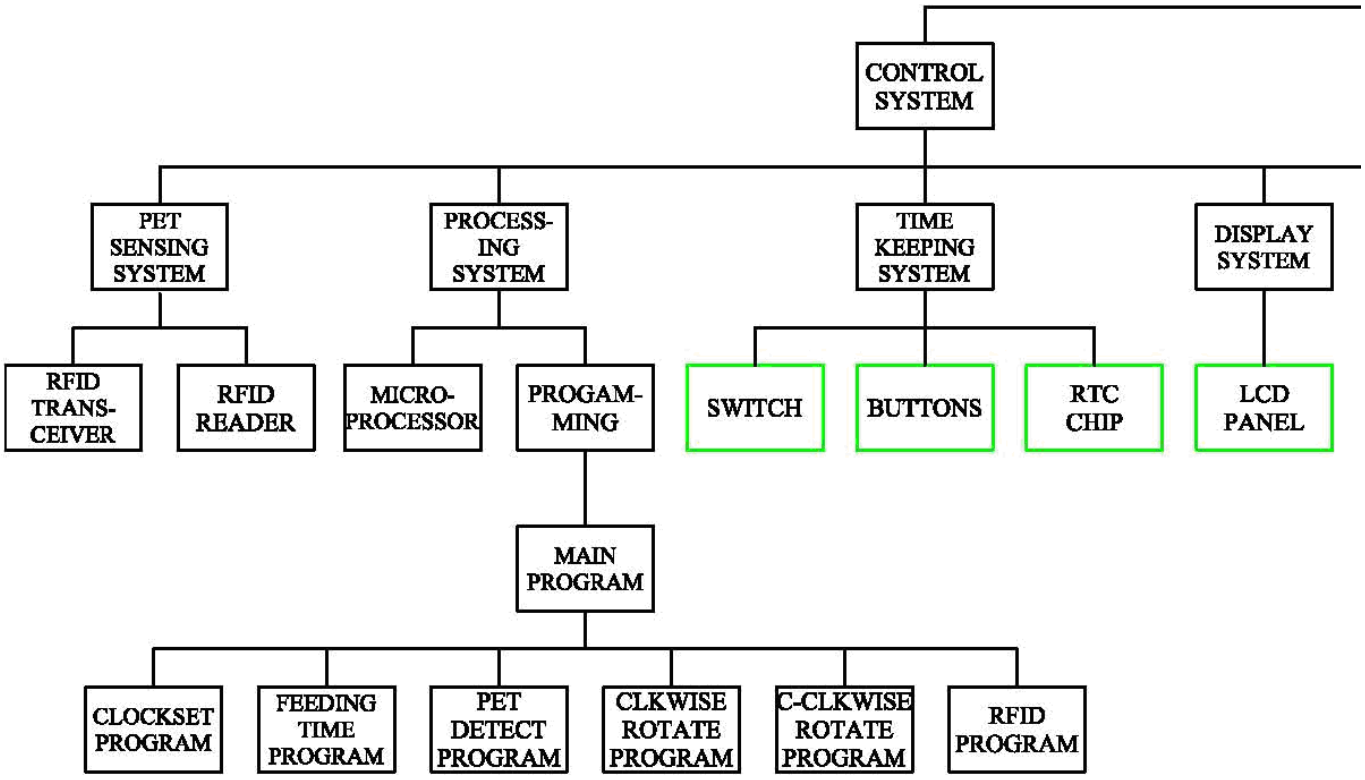


Figure 3: The Control System

The Pet Sensing System:

(Primarily responsible team members: Kristine McCarthy, Alexis Rodriguez-Carlson)

The Pet Sensing System fulfills the design objective to create a product that can distinguish between different animals in order not to allow the forbidden pet to eat from the feeder. In order to do this it is necessary for the forbidden pet to wear an emitter on its collar that a sensor on the feeder can detect and react to.

Our requirements for the emitter/sensor pair are:

1. That the emitter be small enough to be worn comfortably on the pet's collar
2. That the emitter does not pose any health risks to the pet
3. That the sensor fit on the feeder
4. That the sensor be able to interface with the microcontroller
5. That the sensor be able to react to the emitter while the pet is still 4-6 inches away from the feeder

We have chosen an RFID Reader / tag combination for the pet sensing system [1]. We have two different types of tags, one of which will be placed on forbidden pet's collar (see Figure 4 and Figure 5). The tag pictured in Figure 4 is passive, and the tag in Figure 5 is active. For a detailed description of the differences between passive and active tags, see [1].

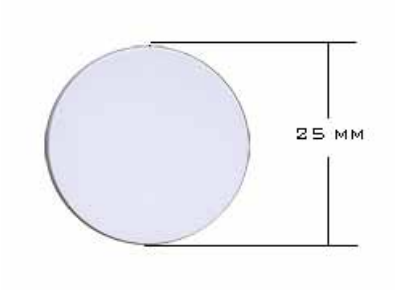


Figure 4: Passive RFID sticker [2]

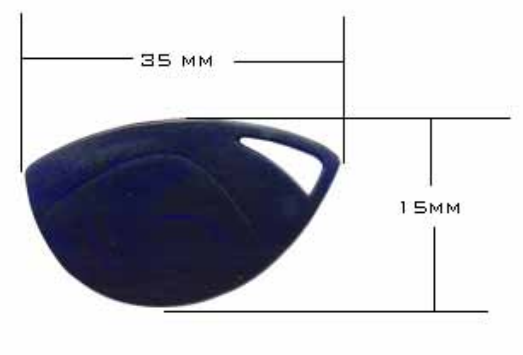


Figure 5: Active RFID Fob [2]

After testing both tags, we have found that the RFID Reader starts responding to the passive tag when it is about 2.5 – 3 inches away and the active tag when it is 4 – 5 inches away. At this time, we are still unclear about whether the extra distance gained by using the active tag is worth the possible health considerations [1]. If we do decide to use the active tag, we will attempt to shield the animal from the radio waves with the help of Professor Badjou.

When testing the RFID Reader, we discovered that we had misunderstood the way the RFID Reader would react to the presence of the tag. We had expected that the Reader would change the logic state of the SOUT pin if any tag were present. In fact, the SOUT pin is high if no tag is present, and if a tag is present, then a "unique ID will be transmitted as a 12-byte ASCII string via the TTL-level SOUT (Serial Output) pin" [2]. For our purposes, we are interested only if any RFID tag is present and not if a particular one is present. Therefore, we have written the code so that it will take a number of readings from the SOUT pin and store those values as variables. Since some of those readings will be zeros, when all of the stored values are ANDed together the result will be zero, which will be stored as variable called "tag". Then, the program only needs to check the condition of "tag" to decide whether the forbidden pet is near or not. This will make the programming both easier to write and faster. Please see the section on software for further details.

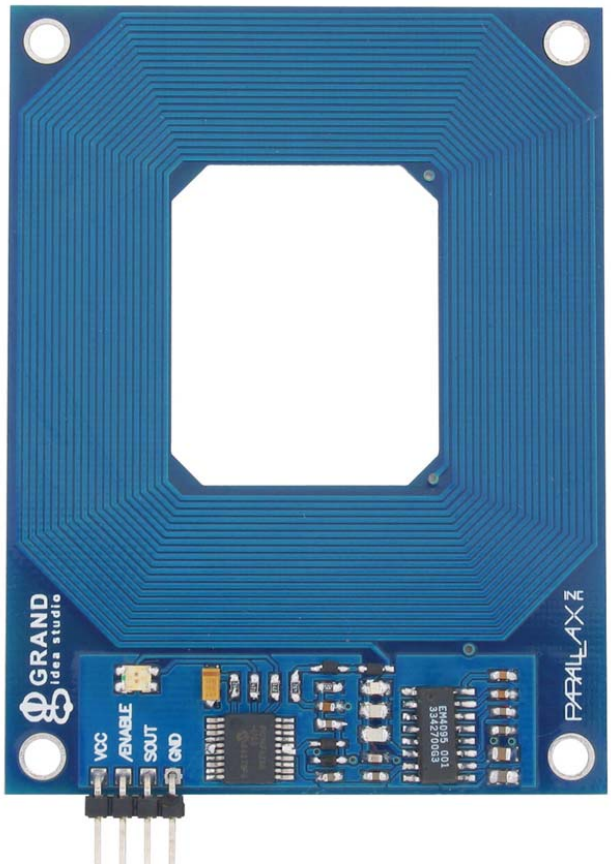


Figure 6: RFID Reader Module [2]

We have also decided that the /Enable pin does not need to go to a data port on the CML12S, and will instead be grounded. This is because the Reader is active when the /Enable pin is low, and we want the reader to be active at all times to provide the quickest response to the forbidden pet's approach.

Now that we have the RFID Reader and tags in hand, we are completely confident that they satisfy all of the requirements of the pet sensing system. Both of the tags are no bigger than any of the other tags typically worn on a pet's collar. We can use either the passive tag or shield the active tag and feel confident that the tag will not have an adverse affect on the pet's health. The RFID Reader is small enough to be mounted on the pet feeder enclosure. We have connected the RFID Reader to the microcontroller and successfully received information from it. Lastly, we have tested the RFID Reader with the tags and learned that while the read range is not as great as we would have liked it to be, it is large enough for our purposes.

The Processing System:

(Primarily responsible team member: Alexis Rodriguez-Carlson)

The Processing System is responsible for the integration and control of the Control System and the Motor System, as well as the different subsystems that make up the Control System. It consists of two parts: a microcontroller and the program. The requirements for a microcontroller are:

1. It must be capable of interfacing with all of the other components that make up the Smart Pet Feeder
2. It must have enough memory to hold the programming required to control all of the components
3. It must have enough RAM to run the program
4. It must be affordable

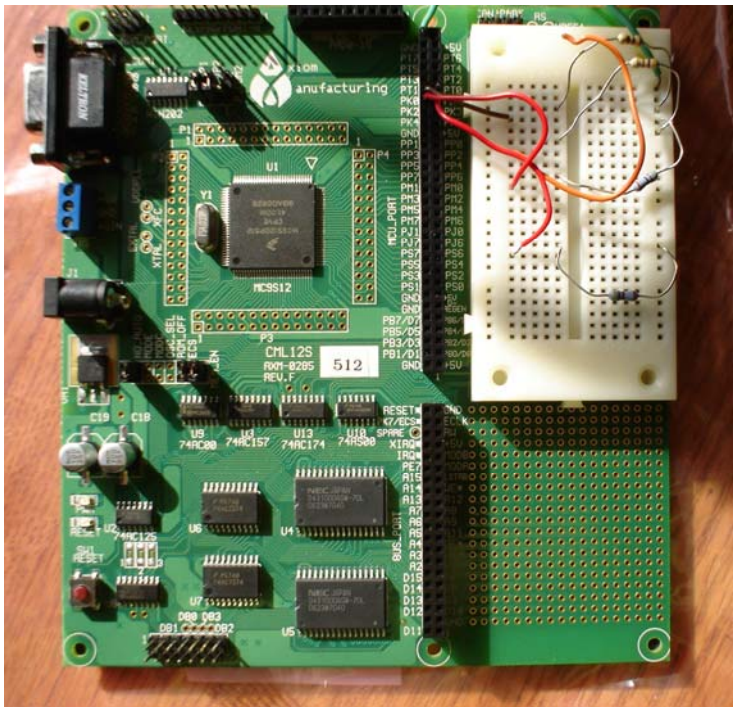


Figure 7: The Axiom CML12S-DP512 Development Board

There have been no changes in our choice or use of the microcontroller [1]. However, there have been some slight changes to the Electrical and Control Schematic (see Figure 8). These changes are:

1. That the resistor and capacitor values for the motor driving circuit have been determined (see the Motor System section for more details)

2. That the /Enable pin of the RFID reader now goes to ground (see the Pet Sensing System section for more details)
3. That all of the lines which had been going to Port B are now in Port T (see the section on the Time Keeping System for more details)

These changes have helped us to fulfill the requirements for the microcontroller as discussed above. We did have a slight hiccup using the CML12S. When downloading some programs, the dotted line, which traveled across the terminal window to symbolize the progress of the download, would stop and the terminal window would freeze. We thought this meant that the program was not downloading, but in fact the program was downloading and the microcontroller was just not returning a "done" statement to indicate that the transfer was successful, even though it was. To run the program we just had to restart the CML12S and enter the "go 4000" command, and the program ran fine.

It will be the Main Program that determines the operation of the feeder. This program will initially determine if the override switch is on or off. Based on the result of this, the program will determine which subroutine to call. If the override switch is on then the program will run the Clockset subroutine. If it is off then it will run the pet detect subroutine. The main program is also where all functions will be declared, as well as any variables that need to be declared outside of their individual functions. Lastly, the main program is where all of the input and output ports that are used will be configured as inputs or outputs as specified in the control schematic. For more details, please see the Program section of the report, as well as Appendix B.

The CML12S continues to satisfy all of our criteria for the microcontroller. It interfaces with all of the other components that make up the Smart Pet Feeder. It has ample memory to hold the programming required to control all of the components and more than enough RAM to run the programs, and has not required us to spend any money to make it work properly.

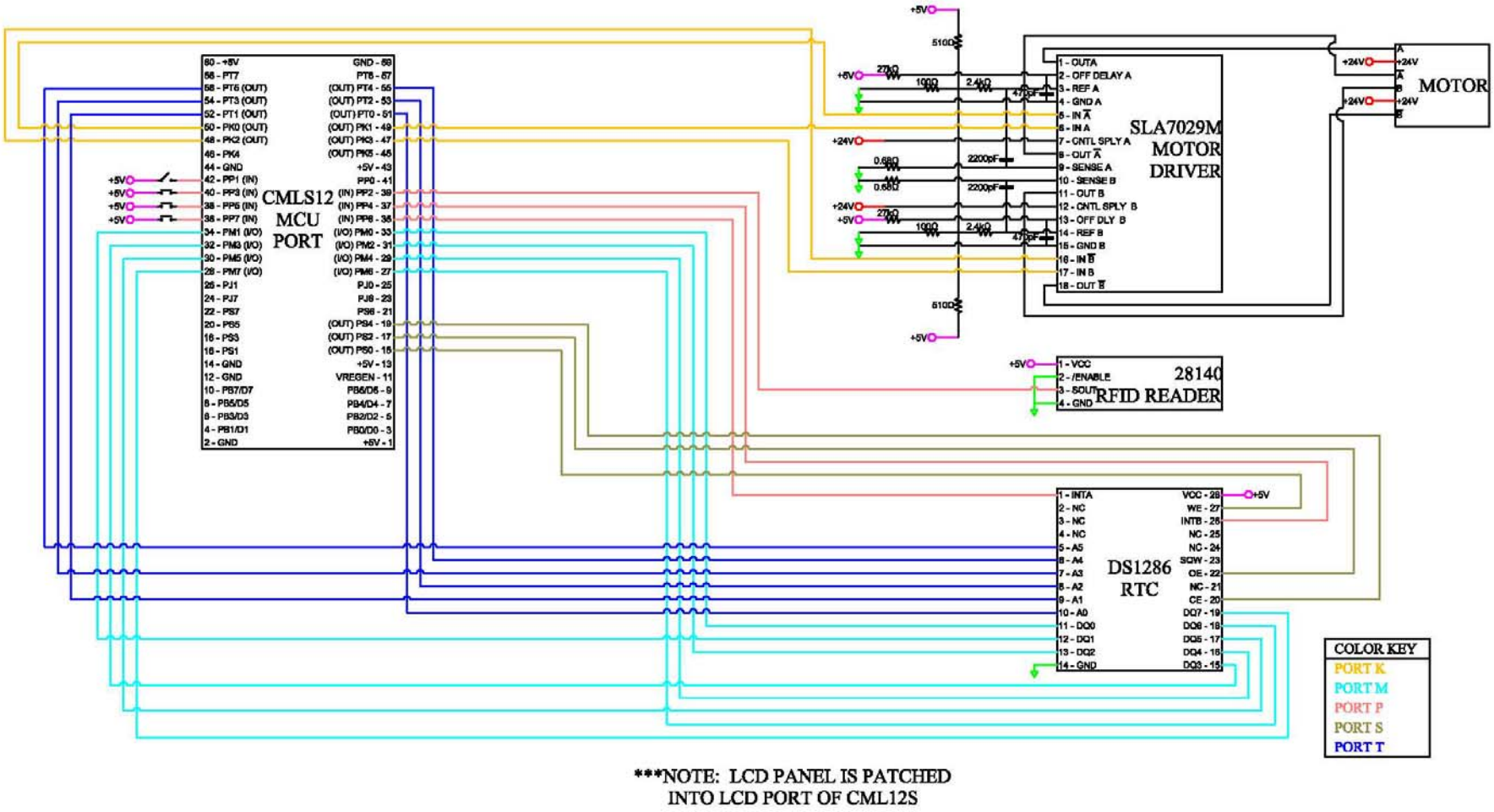


Figure 8: Electrical and Control Schematic

The Time Keeping System:

(Primarily responsible team member: Kristine McCarthy)

One of the subsystems of the Control System is the Time Keeping System. This is the portion of the design that will keep time, allow the user to program when the feeder should rotate in order to reveal a fresh bowl of food, and provide incremental timing. The criteria that this system must meet are:

1. To allow the user to set the time on the clock
2. To tell the feeder to rotate at specific times during the day to reveal fresh food
3. Use a toggle switch as a manual override to prevent the bowls from rotating
4. When the toggle switch is turned off, allow the program to rotate the bowl to the blank spot
5. Provide incremental timing

Since the last reporting period, significant progress has been made in this system. First, we now understand how to communicate with the Real Time Clock (RTC) chip. The DS2186 comes in a 28-pin encapsulated package. In this package, there are six address pins, as seen in Figure 9. These pins allow the user to specify the register of the chip the user is accessing [3]. These pins caused confusion because we did not understand why there were only six pins for the DS2186's 63 registers (see Figure 11).

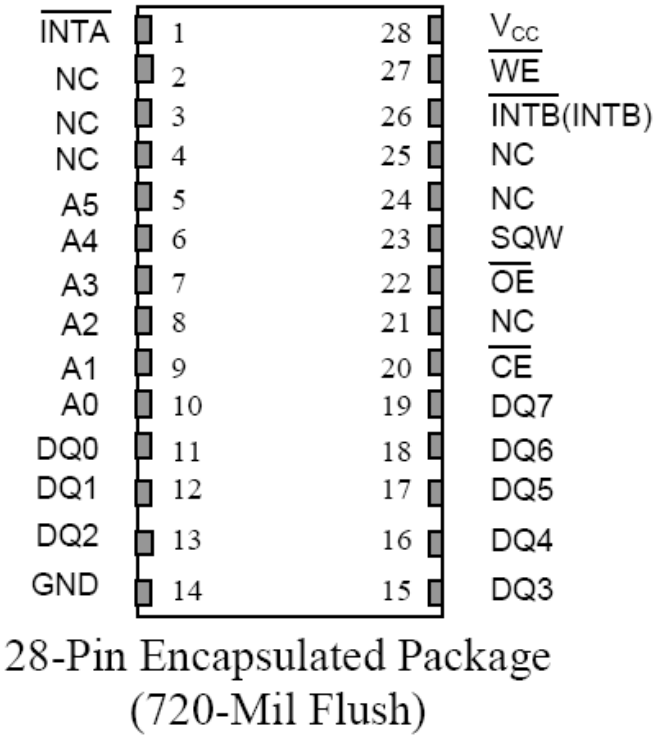


Figure 9: RTC Pin Assignments [3]

After studying the data sheet extensively, we realized that communication with the RTC is always in Binary Coded Decimal (BCD) [3]. BCD is numbering system such as binary and hexadecimal. In this system, each digit of a number is broken down into a four digit binary number and then written as a binary number, hence the name "binary coded decimal." For example, the address of the largest register, as can be seen in Figure 11, is hexadecimal 3F, 63 in decimal, or 00111111 in BCD. This means that you can tell the microcontroller which of the 63 registers to access with only six pins by using BCD. For example, in order to set the hour, the address pins would first need to be set to register 4 (see Figure 11), or 00000100 in BCD, then programming can begin.

Conversion Code - Chart																
DECIMAL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
BINARY	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Figure 10: Decimal to Hexadecimal to Binary Conversion Chart [4]

In order to program the chip, several things need to happen. As stated, the register needs to be specified on the address pins, but even before that, the output enable, chip enable, and write enable pins need to be

configured correctly. There are separate configurations for reading and writing. In order to write, the write enable and chip enable pins must be in the active, or low, state. Then the data that is to be written to the chip (time) is input via the data input/output pins [3].

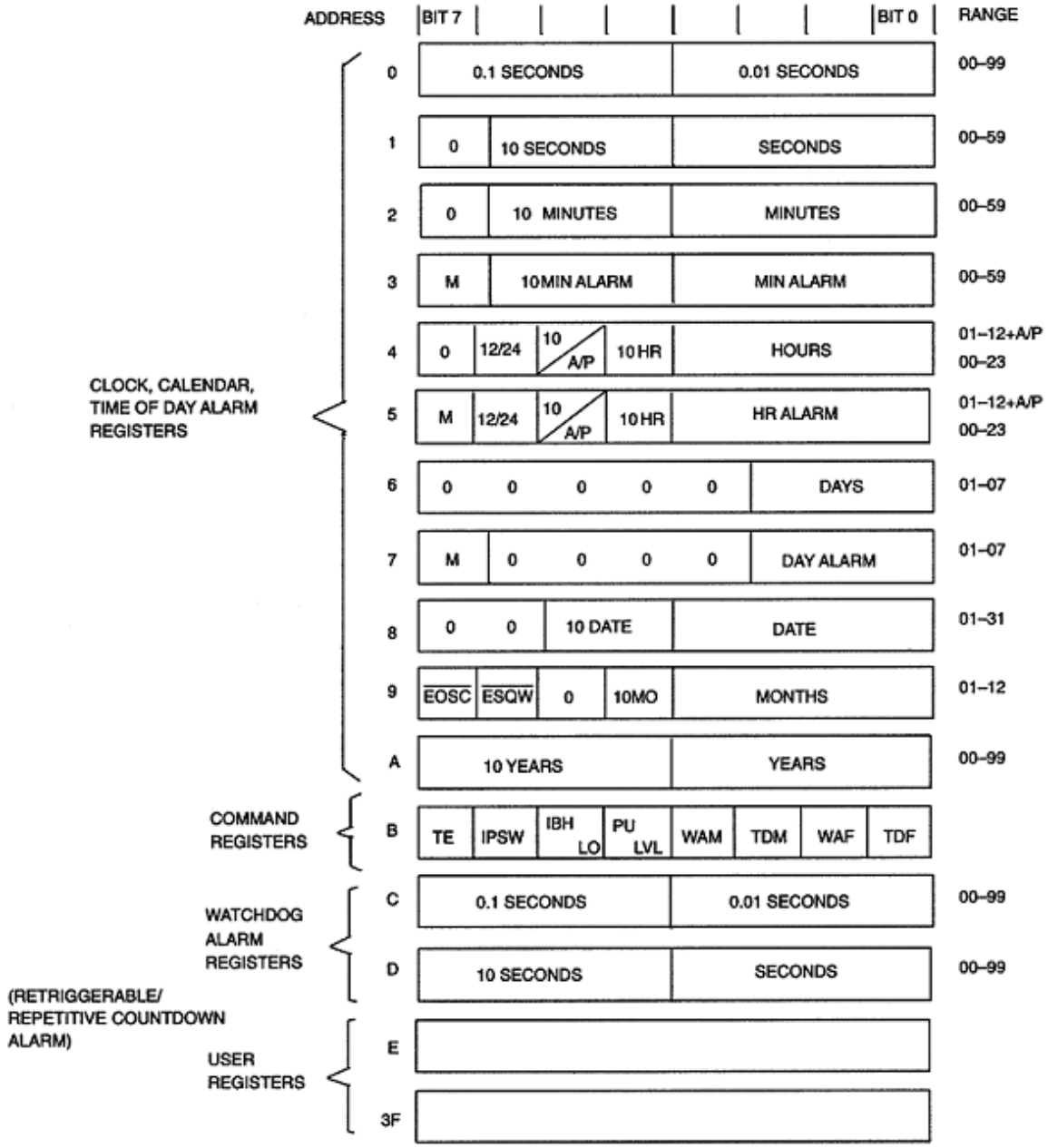


Figure 11: RTC Registers [3]

Another thing that we have accomplished with the RTC is to understand how the Watchdog Alarm works. This feature acts as a counter that is capable of counting down a maximum time of 99.99 seconds. In

order to use this feature, all you have to do it is set the Watchdog Alarm registers, C and D, to the amount of time that you want to be counted down [3]. In our case, this is 30 seconds and will be used after the RFID has sensed that the forbidden pet has approached the feeder and the motor has rotated the bowl to the blank spot. When the countdown reaches zero, the Watchdog Interrupt Output goes into the active state and sends a signal to Interrupt A (INTA) of the chip [3] which can also be seen in Figure 11. The other interrupt (INTB) is used for the time of day alarm. It is utilized when the values in the alarm registers match the values in the time registers [3]. Both interrupts will then be sent to Port P on the CML12S where they can be used in other programs. An interrupt is sent when the interrupt output of the chip enters the active state, which sends a high signal [3] to the pin on the CML12S; we have chosen to use Port P for this purpose. The pin that the interrupt is connected to will then go from a logic 0 to a 1 indicating the interrupt has occurred.

As mentioned earlier, we are going to use the RTC's counter feature. We had believed that a program was going to be necessary to control its operation, but after having rethought how the Watchdog Alarm works, we have determined that it is unnecessary. We realized that after the delay is programmed into the RTC, the chip continually counts down to 0 and sends a signal to let the chip know that an alarm has occurred, so there is no need for coding [3].

So far, this information only covers how the chip is accessed by our program, not how the user will be able to set the clock and meal times. This will be accomplished using three buttons. These buttons will have different functions depending upon what portion of the program the user is in. For more specific details, please see the Program Section of the report.

One problem we encountered when performing tests with the RTC, was with addressing Port B on the CML12S, namely. For some reason, we were able to communicate with all of the ports except Port B. All of the ports and individual pins on the MCU are labeled next to them on the board, and there are two separate designations next to Port B (see Figure 12). We are not exactly sure what this extra designation is so we have decided, in the interest of time, to forego the use of that port and move these connections to Port T, as it was not being used. This change can be seen in the updated control schematic.

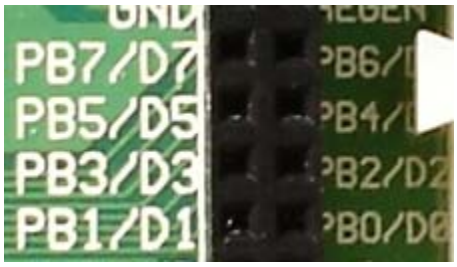


Figure 12: Port B of the CML12S

Further understanding how to use this RTC chip and its incorporation into the Time Keeping System helps us to realize our design objectives. This RTC allows us to keep the time and to use two separate alarm features to reveal a fresh meal at a user programmed time and to allow for a delay in the RFID reader program. This delay will allow time for the forbidden pet to move away from the bowl before the program checks to see if the pet is still there. It also allows the user the freedom of easily changing the time of day or rotation time without needing to change software programming or hardware.

The Display System:

(Primarily responsible team member: Rachel Heil)

The main display system is an Liquid Crystal Display (LCD) panel, which will allow the user to interface with the CML12S in order to set the clock and the meal times.

The requirements for the LCD panel are as follows:

1. That it is cost effective
2. That it is compatible with the CML12S microcontroller
3. That it be capable of displaying the time and text necessary for setting the feeder

Figure 13: The selected LCD panel and its connector.

The LCD panel that was selected was inexpensive and it was created to work with the CML12S [1]. This addresses the first two requirements above. Since the last progress report, we have successfully established communication between the CML12S and the LCD panel, and displayed text on all four lines of the LCD. This was accomplished using free code provided by Axiom (see Appendix C). This means that we now have the ability to program what we would like to appear on the LCD panel, satisfying requirement number 3.

The main problem we encountered with the LCD panel was creating the programming necessary to display text to the LCD. After a couple weeks of the researching and trial and error, we contacted the manufacturer, Axiom, and were provided with guidance and working freeware code including an LCD driver and an LCD test (see Appendix C). We were still not sure exactly how to use it; we repeatedly received compilation errors when we tried to compile either of the programs. We found that we needed to include both of the code files within the same project in order for it to compile correctly. Originally, we were using the test program only, and would receive errors indicating that that the functions in the program were undefined. The second program that needed to be used is where these functions were defined. After compiling the project with both files included we received no errors, and the designated text was displayed on the LCD screen.

The LCD panel now meets all of the necessary requirements. It is cost effective and compatible with our microcontroller. In addition, we now have full communication with our LCD display panel and are able to use the previously mentioned programs in order to tell the LCD exactly what text to display.

The IDE:

(Primarily responsible team members: Rachel Heil)

The Integrated Development Environment (IDE) is the software in which we will write and compile the code that will tell the CML12S what to do. The requirements for the IDE are as follows:

1. That it is cost effective
2. That it allows each team member to have a copy of the software in order to program simultaneously
3. That it be compatible with the HC12 microcontroller
4. That it allow us to program using the C language
5. That it be easy to use

Originally, we had chosen to use Embedded GNU for our IDE [1]. The reason we changed our IDE is that we needed administrator rights on our computers in order to compile a program. This became an issue because most of the team members are working on laptops, which do not have the necessary rights. For this reason, we decided to switch our IDE. The alternative we chose to use is ImageCraft's ICCV7 for CPU12.

ICCV7 was selected for a few reasons. The first was that it was created to operate with the HC12 family, which includes the CML12S that we are using, so we knew that it is compatible. Also, since we are within the last 45 days of the semester, we are able to use the free downloadable 45-day trial version, allowing each of us to have a copy of the software on our individual computers. This allows for simultaneous programming as per requirement number two. The IDE allows programming in the C language, which makes for easier programming. Another advantage to using the ImageCraft software is that we have used the ICC11 version of the software previously and know the environment, which means that there is minimal time spent on learning the IDE itself.

The ICCV7 was chosen because it meets all five of our requirements. It is cost effective, compatible with our microcontroller, and available to all members of the group at the same time. In addition, the IDE

allows for programming in the C language and is an environment we are familiar with and therefore is easier for us to use.

The Program:

(Primarily responsible team members: Rachel Heil, Kristine McCarthy, Filip Rege, Alexis Rodriguez-Carlson)

The requirements for the program have changed slightly in the past weeks as we have begun to write code and become more familiar with our components. These changes are reflected in the list of requirements below. The program must:

1. Control the microcontroller and all of the components (Main Program see Appendix B on page 52)
2. Provide a means for the user to set the time (Clockset Program)
3. Provide a means for the user to program the time that the motor will rotate to reveal fresh food (Clockset Program)
4. Provide a way for the user to suspend operation while refilling the food dishes (all Programs)
5. Keep track of which dish is revealed so that the system knows how far it must rotate to reveal the blank spot (Feeding Time Program see Appendix D on page 49)
6. Monitor the RTC chip to see if it is sending an alarm signal (Feeding Time Program)
7. Monitor the RFID reader to see if it is receiving a signal from the tag (Pet Detect Program see Appendix E on page 49; RFID Program see Appendix A on page 49)
8. Provide incremental timing
9. Rotate the motor both clockwise and counter clockwise a specified distance (Motor Rotation Program see Appendix F on page 66)

The algorithms and flow charts which have been created since the last progress report are below.

Program Algorithms:

RFID Program:

1. Set DDRP = 0 to configure Port P as in input

2. Set variable A-R equal to the value of Pin 2 of Port P
3. Set TAG equal to variables A-R ANDed together
4. If TAG = 0
    a. PET = 1
5. If TAG = 1
    a. PET = 0
6. Return PET to main program

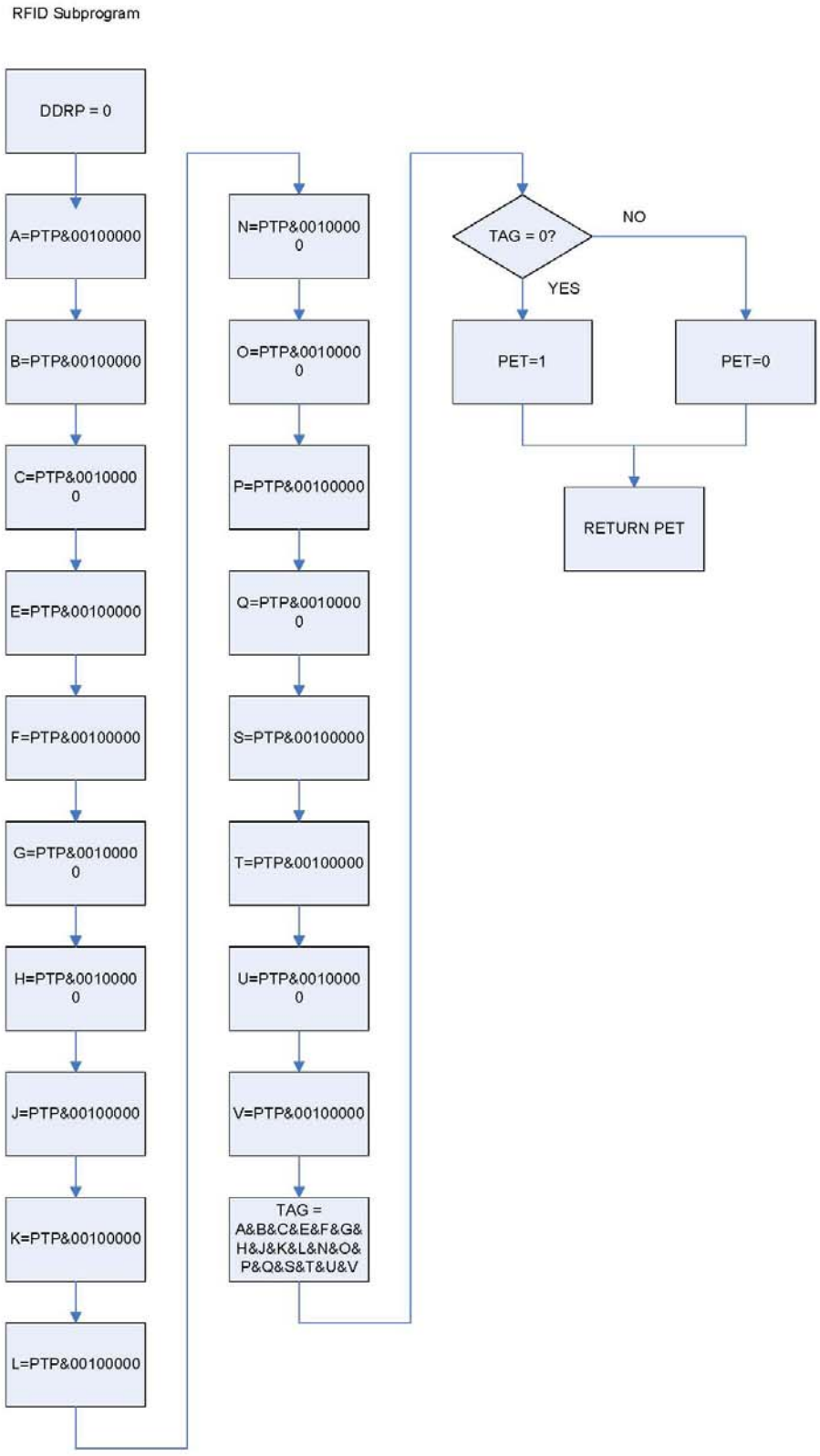


Figure 14: Flowchart of RFID subprogram

Clockset Program:

So far, there is one program that will be affiliated with the RTC. This program is the Clockset Program. This is the program that allows the user to set the clock and meal times and then rotates the tray back to the blank spot of the feeder.

Variables: TH, TM, AH, and AM are Time Hour, Time Minute, Alarm Hour, and Alarm Minute respectively.

1. Display greeting “To reset time press button 1, to reset alarm press 2, to exit time set turn the override switch off.”
2. IF OVERRIDE = 1
    a. IF Button 1 = 1
        i. Display time
        ii. Display “To change hour press button 1, to change minute press button 2, to end, press button 3”
            1. IF Button 1 = 1
                a. inc TH
                b. update WTC
                c. update display
            2. IF Button 2 = 1
                a. inc TM
                b. update WTC
                c. update display
            3. IF Button 3 = 1
                a. go to Line 1
            4. IF Button 2 = 1

iii. Display alarm time

iv. Display “To change hour press button 1, to change minute press button 2, to end, press button 3”

1. IF Button 1 = 1
    a. inc AH
    b. update WTC
    c. update display
2. IF Button 2 = 1
    a. inc AM
    b. update WTC
    c. update display
3. IF Button 3 = 1
    a. go to Line 1

3. IF OVERRIDE = 0
    a. Rotate counter clockwise BOWL x SLICE (Returns to blank spot)
    b. Call Pet Detect

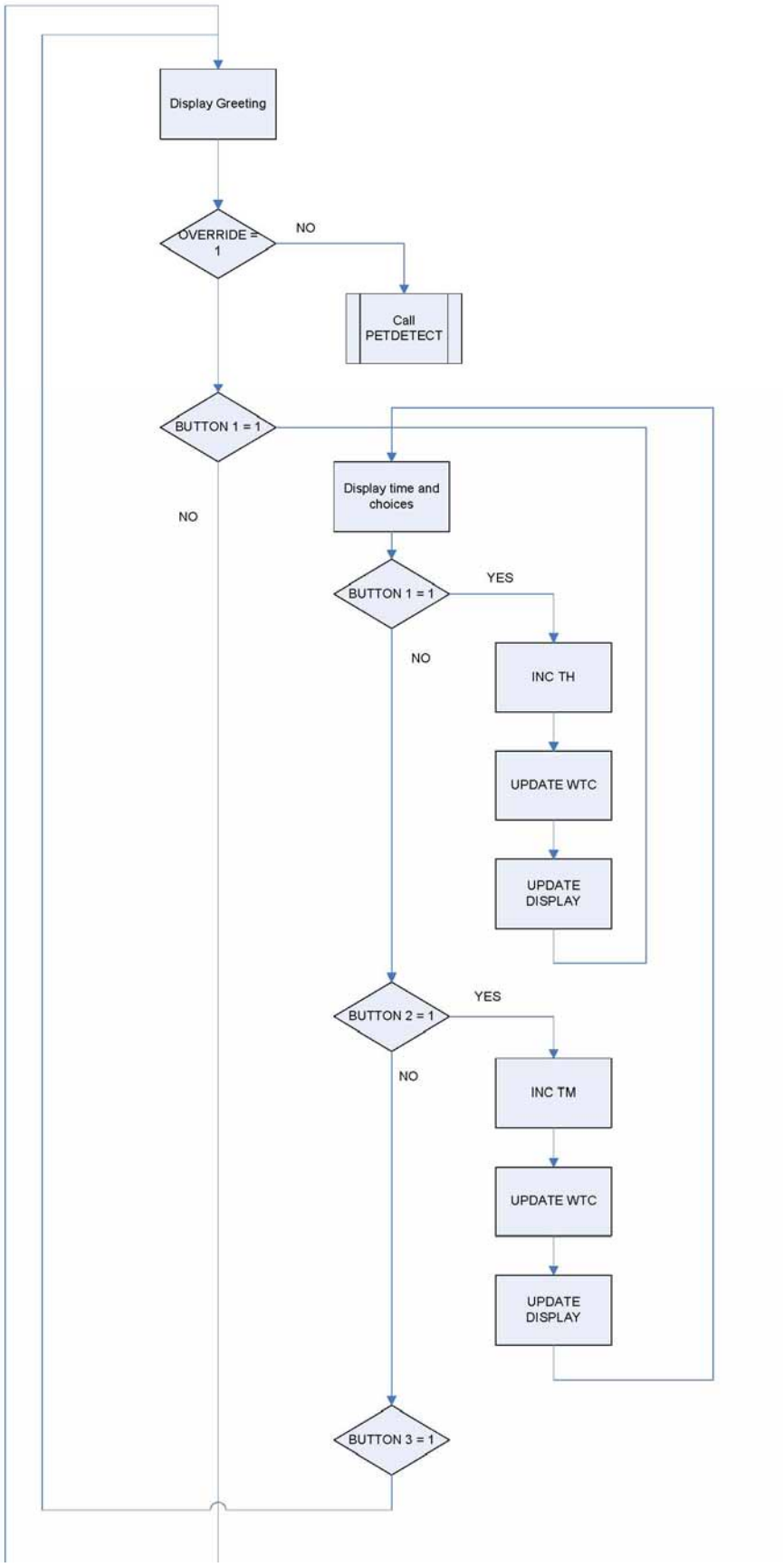


Figure 15: Flowchart for Clockset Program Part 1

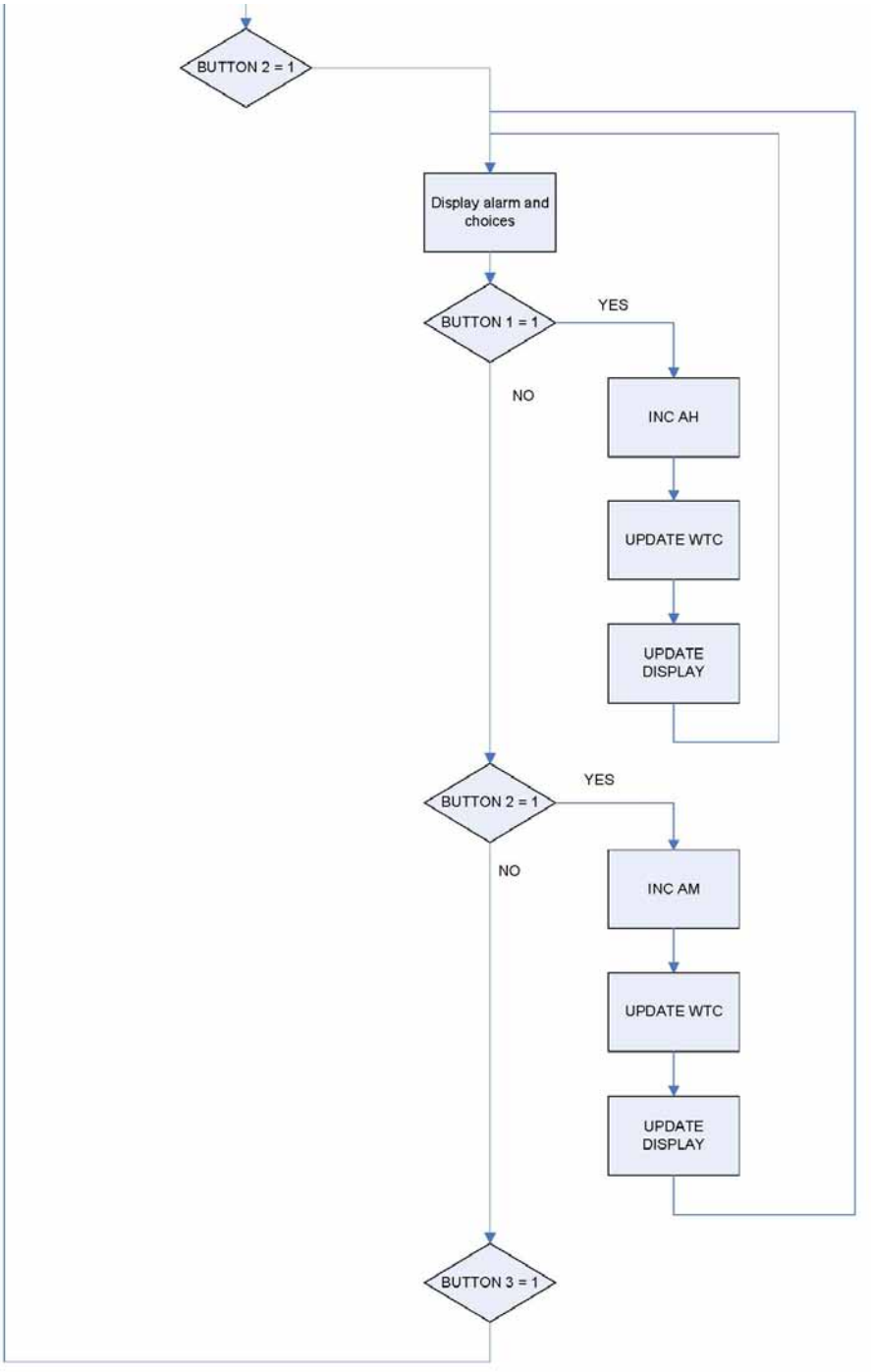


Figure 16: Flowchart for Clockset Program Part 2

Motor_CW:

NUM_OF_STATES – eight different styles that are put in Port K

DELAY_MAX – maximum number of counts to create a delay

i – an integer variable that is used in the *for* loop that creates a time delay before

state_array – a character variable that stores the state valuse that will be input in Port K

steps_to_move – an integerr variable that stores the number of steps that the motor will move

next_state – character variable that selects the next state to put in Port K

1. Set DDRK = 11111111 to configure it as an output
2. Set Port K = 00000000
3. Set steps_to_move to the desired number of steps
4. Set next_state = 0000
5. Set Port K = next_state
6. Set i = 0, check if i < DELAY_MAX
    a. If it is, increment i by 1 and go back to step 6
    b. If it is not, proceed to the next step
7. Check if steps_to_move > 0
    a. If it is, check if next_state > (NUM_OF_STATES -1)
        i. If it is, set next_state = 0
    b. Otherwise proceed to the next step
    c. Set Port K = state_array[next_state]
    d. Set i = 0, check if i < DELAY_MAX
        i. If it is, increment i by 1 and go back to 7d
    e. Increment next_state by 1
    f. Decrement steps_to_move by 1
8. End Main

NUM_OF_STATES – eight different styles that are put in Port K

DELAY_MAX – maximum number of counts to create a delay

i – an integer variable that is used in the *for* loop that creates a time delay before

state_array – a character variable that stores the state valuse that will be input in Port K

steps_to_move – an integerr variable that stores the number of steps that the motor will move

next_state – character variable that selects the next state to put in Port K

9. Set DDRK = 11111111 to configure it as an output
10. Set Port K = 00000000
11. Set steps_to_move to the desired number of steps
12. Set next_state = 0000
13. Set Port K = next_state
14. Set i = 0, check if i < DELAY_MAX
    a. If it is, increment i by 1 and check again
    b. If it is not, proceed forward
15. Check if steps_to_move > 0
    a. If it is, check if next_state < 0
        i. If it is, set next_state = (NUM_OF_STATES – 1)
    b. Set Port K = state_array[next_state]
    c. Set i = 0, check if i < DELAY_MAX, if it is increment i by 1
    d. Decrement next_state by 1
    e. Decrement steps_to_move by 1
16. End Main

The Motor System:

(Primarily responsible team member: Filip Rege)

As Figure 17 shows, the motor system consists of the following components: stepping motor, motor driver, power supply and tray support.

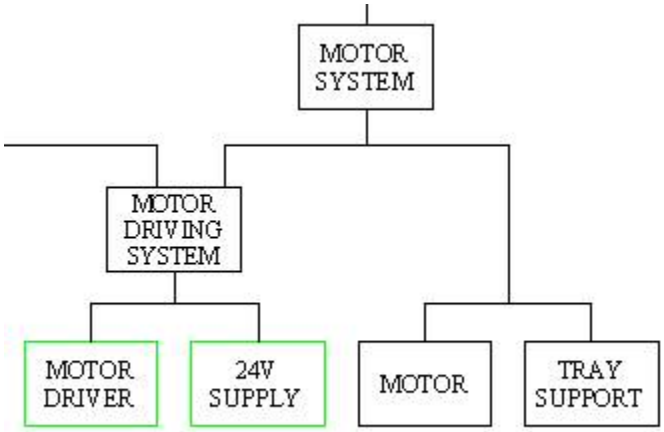


Figure 17: Block Diagram of the Motor System

The Motor Driving System:

Since the last progress report, a lot of progress has been made in the design of the circuit necessary to use the driver chip. A driver chip is an integrated circuit (IC) that serves two purposes: it controls the flow of the current through the motor and protects the microcontroller from the motor's high current. Different types of drivers exist and one needs to select the appropriate chip depending on the type of motor and its power rating. The driver chip we chose to use had to meet the following criteria:

1. It had to be capable of handling the voltage (24V DC) and the current (1.5A) necessary for the motor's operation [5]
2. It had to be easily interfaced with the microcontroller
3. It had to cost less then $10.00

We have chosen the SanKen manufactured SLA7026M (Figure 18) driver chip. It is a good choice to go with our motor (the 4118M-06 [1]) because it meets all the requirements. It is capable of handling up to 3.0A

of current and up to 46VDC [6], which is more than enough considering that the motor draws only 1.5A at 24VDC [5] .

Figure 18: SLA7026M Driver Chip

The CML12S can neither supply nor sink more than 25mA of current [7]. The motor, however, needs at least 1.0A to operate [5]. Therefore, not only must the current to the motor be delivered from a source other than the CML12S, but also the microcontroller needs to be protected from the high current of the motor. This is one of the two functions of the driver circuit, the other being energizing and de-energizing the coils in the motor in a sequence dictated by the microcontroller. The driver may be thought of as a hub to which the motor, the power supply, and the microcontroller are connected. It determines what signals may pass and where they go.

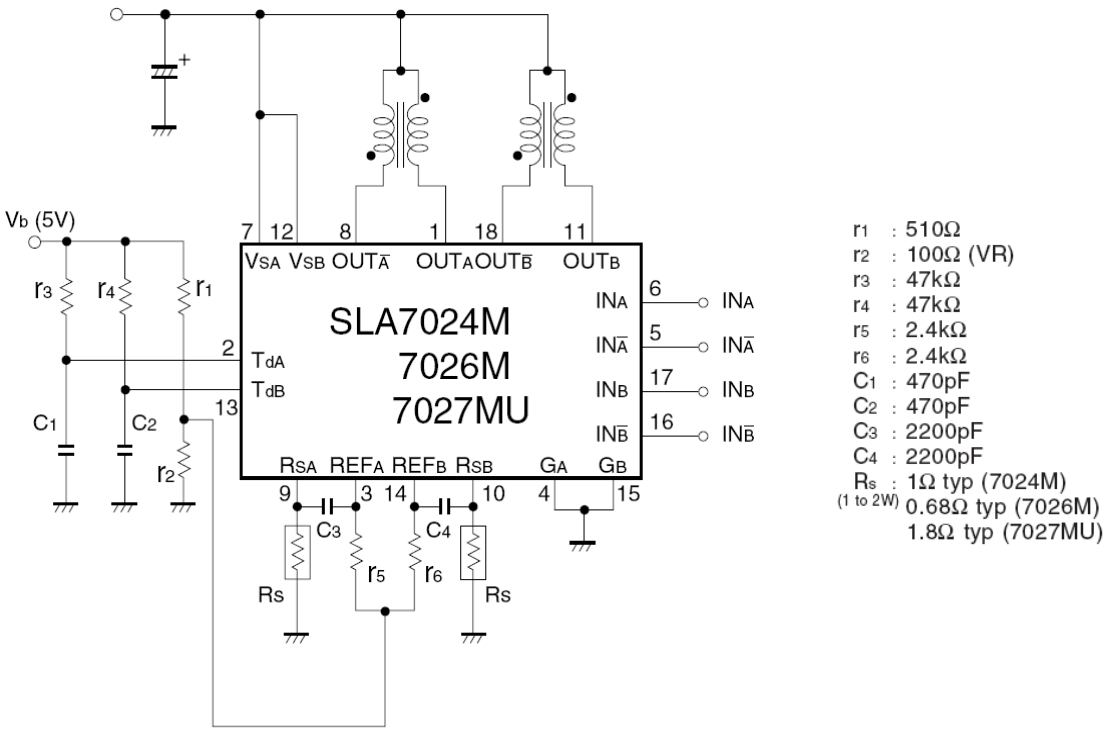


Figure 19: Motor Driver Circuit [6]

The main obstacle we have encountered with the driver chip was to determine the values of the resistors and the capacitors that connect the SLA7026M to the ground on one side and to the microcontroller on the other side (Figure 19). SanKen, the chip manufacturer, provides a list of recommended values in the datasheet but we were not sure whether these needed to be recalculated to match our specific power requirements. Finally, with the help of Joseph Diecidue and professor Badjou, we were able to determine that the recommended resistors and capacitors were accurate as long as the current was less then 3.0A.

Another problem was that the pins of the SLA7026 are not compatible with the breadboard. They are not only wider than the holes in the breadboard but also spaced out in way that does not align with the grid of the breadboard holes. We solved this problem by attaching wires to all the pins, which allowed us to connect the chip to our breadboard. As of March 28, we have accumulated all the components for the driver circuit and started assembling it.

The driver chip is the hardware which interfaces the CML12S to the motor. The CML12S will control the motor through two programs, one which will turn the motor clockwise a distance to move to the next bowl

and another which will turn the motor counter clockwise the same distance. The program will accomplish this by powering on the magnets in the motor in order for a specified length of time. When the tray needs to turn a longer distance, the program will run multiple times. Please see the program section for more information.

The Tray Support System:

The function of the dish tray support system is to isolate the weight of the tray with the dishes and pet food, a maximum of two pounds, from the motor's shaft. Stepping motors are designed to carry torsional loads but not axial loads. In other words, they are not equipped to withstand loads that either pull or push on the shaft. A model of the Tray Support System is shown in Figure 20.

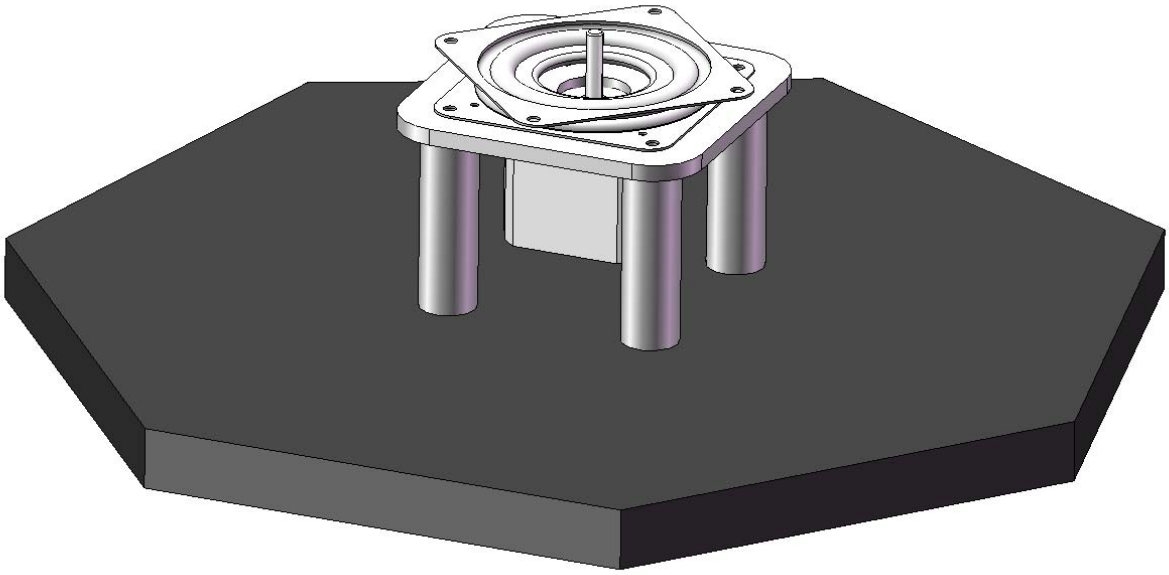


Figure 20: Tray Support System mounted to the Feeder's Base

The central piece of the load bearing system is a 3.8"x 3.8"x 0.2"plate machined from 6061 aluminum alloy. The rotary table that will interface between the dish tray and the support system is mounted on top of this plate (Figure 20). As can be seen from Figure 20, the motor is attached to the bottom of the plate so that its shaft protrudes through the center of the plate and the center of the rotary table. Both the shaft of the motor and the hole in the center of the dish tray are keyed to ensure secure fit without slipping. The plate is attached to the

base of the feeder with four bolts. In order to make sure that the bearing plate is parallel to the base, four glass-filled Delrin tubes of uniform length fit over the bolts to support the plate in each corner.

As of the weekend of March 28, all the parts of the support system have been acquired, including the hardware, and the bearing plate has been machined from 6061 aluminum alloy. The only thing that is left to do is to assemble the system.

The Feeder Enclosure:

(Primarily responsible team member: Filip Rege)

The main function of the feeder enclosure is to provide protection for the electronics inside the feeder and to prevent the pet from accessing the food stored for later feedings. In order to achieve these goals it has to:

1. Be strong enough to withstand the weight of the pet, should the pet stand on it
2. Be capable of preventing the pet from accessing food stored for later meals
3. Provide access to the food at the same location every time
4. Be heavy enough to prevent the pet from turning it over
5. Have a removable cover so the user can easily access the bowls
6. Have easily removable dish-washer safe bowls

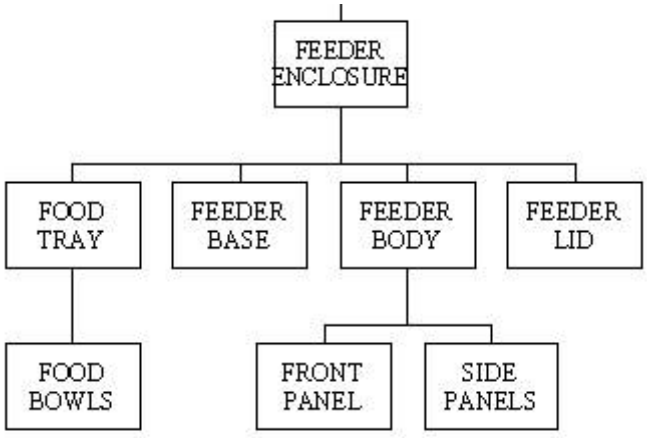


Figure 21: Block Diagram of the Feeder Enclosure Subsystem

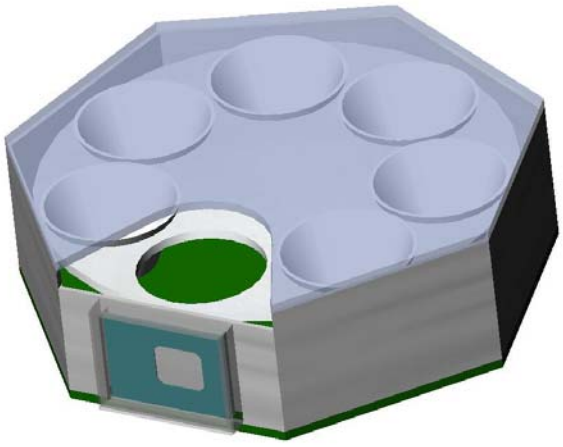


Figure 22: Model of the Smart Pet Feeder

The only change to the feeder enclosure system since the last reporting period is that the base has been machined (see Figure 23). We chose to use polyvinyl chloride (PVC) because it has relatively high density, which will help to keep the product's center of gravity low to the ground. The other reason we chose this material was that we had access to some scraps and, therefore, it did not add any additional cost to our budget. Another interesting property of PVC, that we discovered just recently, is that it is static dissipative so we do not have to worry about a charge building up on the feeder base that might pose a potential danger to the electronic components.

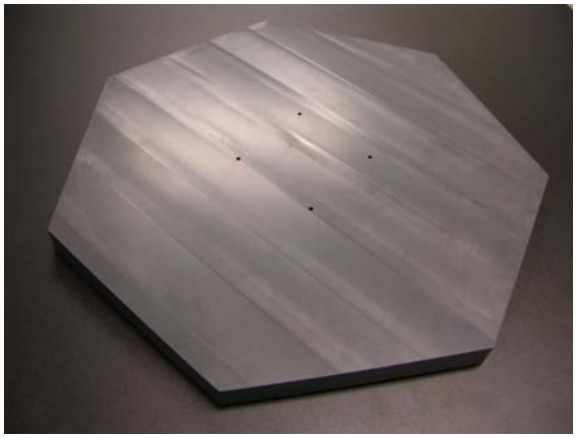


Figure 23: Feeder Base machined from gray PVC

Within the time that remains to the project's deadline, we are planning to build the top and the sides of the enclosure (Figure 22). Currently, we intend to use either a sheet of aluminum or a sheet of stainless steel and wrap it around the heptagonal-shaped base. Also, we are planning to use a 3/16" thick sheet of transparent acrylic for the top of the feeder so that the user can easily see how much food there is left in the feeder and plan to refill it accordingly. The lid will be attached to the rest of the feeder with a few thumbscrews that will make it both easy for the user to remove it and highly unlikely for the pet to pull it off, thus gaining access to all the food inside.

Problem Areas:

In the past month, we have encountered several problems. These problems have been discussed in more detail in the sections above, and have included issues using our original IDE, problems downloading programs to the CML12S, problems getting the LCD programs to compile, confusion about how to use the RTC, and confusion about the values of the resistors and capacitors in the motor driving circuit.

The original IDE we choose appeared to meet all of our requirements, but then it turned out that it would not work properly on our laptops, where we do not have administrative rights. We found that we were within 45 days of the end of the project and so we downloaded ImageCraft's ICCV7 to use as our compiler. For a while, it seemed that none of the programs compiled with ICCV7 were downloading to the CML12S, but it turned out that we were simply misunderstanding the process to use that program. At this time, we have successfully downloaded and run several programs, including the programs necessary to use the LCD panel.

We continue to make progress in our quest to use the RTC as described above. There is still a lot to figure out, but we are confident that we will be able to program it to perform all of the functions we need from it.

Lastly, we have determined the correct values for the capacitors and resistors in the motor driving circuit and have begun to build it. We hope to see the motor turn in the coming week.

Plans for the Next Reporting Period:

We plan to finish the prototype of the Smart Pet Feeder during the next reporting period. This involves:

1. Machining of the remaining parts
2. Building of the tray support system
3. Putting the enclosure together
4. Finalizing and debugging the programming
5. Assembling the complete system
6. Lab testing of the system
7. Animal testing of the system

For details on the tasks to be completed in the next month, please see the Gantt Chart below.

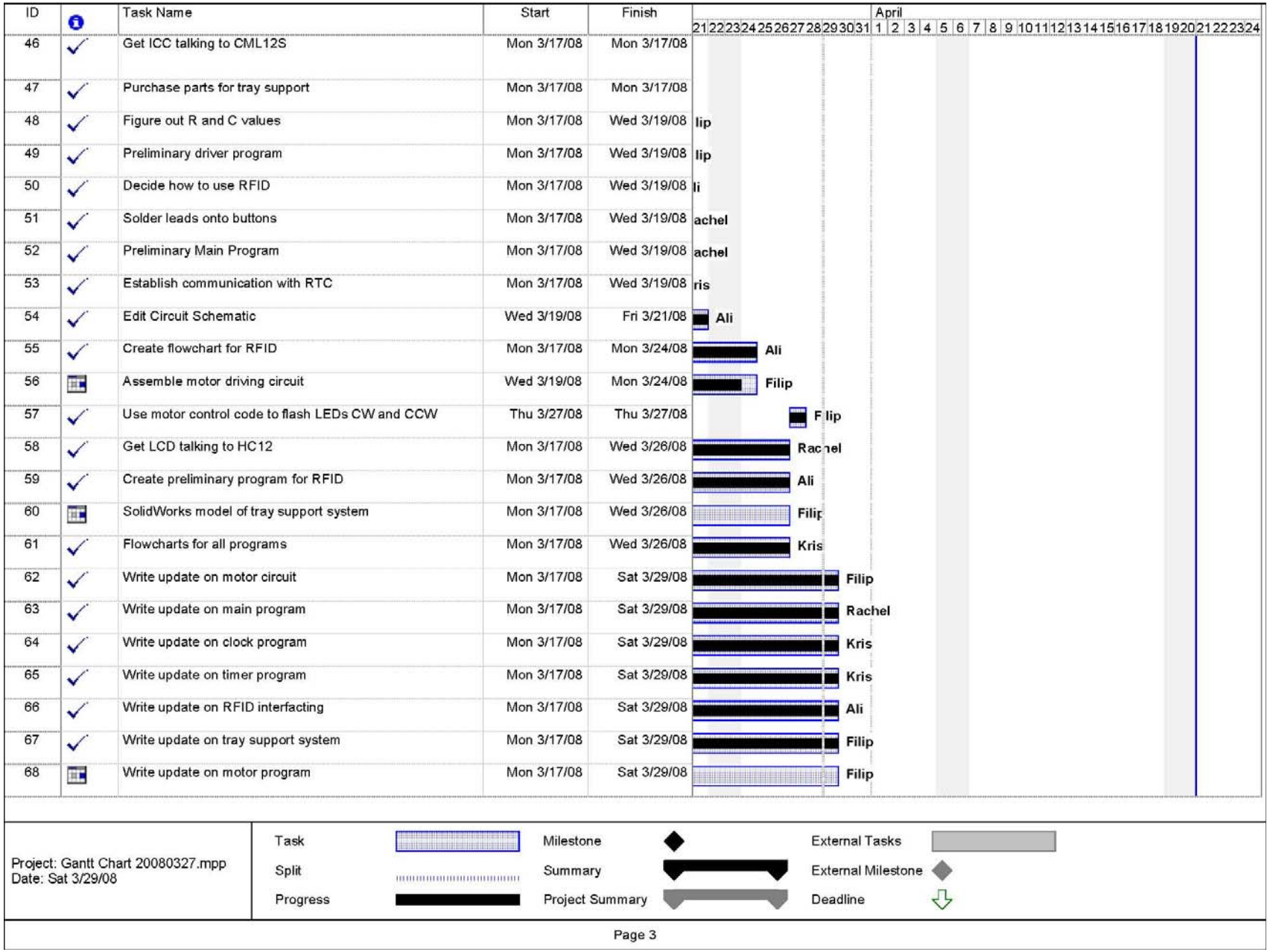


Figure 24: Gantt Chart for the Rest of Project Period Part 1

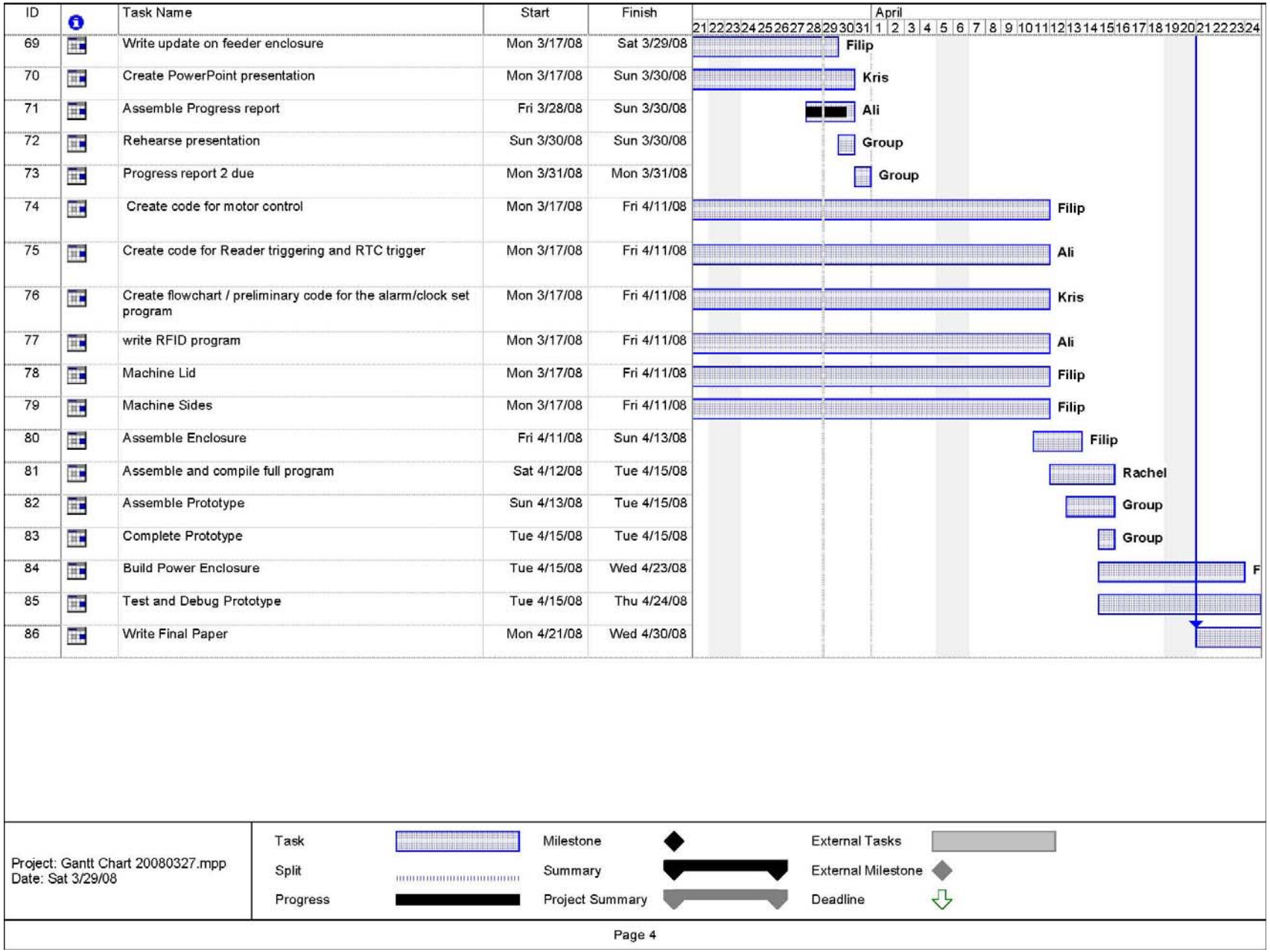


Figure 25: Gantt Chart for the Rest of Project Period Part 2

Schedule Status:

At this time, we are slightly behind schedule in the programming of the RTC and the testing of the motor. This is due to the issues we had getting the CML12S to communicate with our different computers, the complexity of the motor driving circuit, and the fact that, having access to only one microcontroller, each person has limited time to refine the program they are writing. Though we had hoped to be further along in this process, but are still confidant that we can finish the project on schedule. For more details, please see the Gantt Chart below.

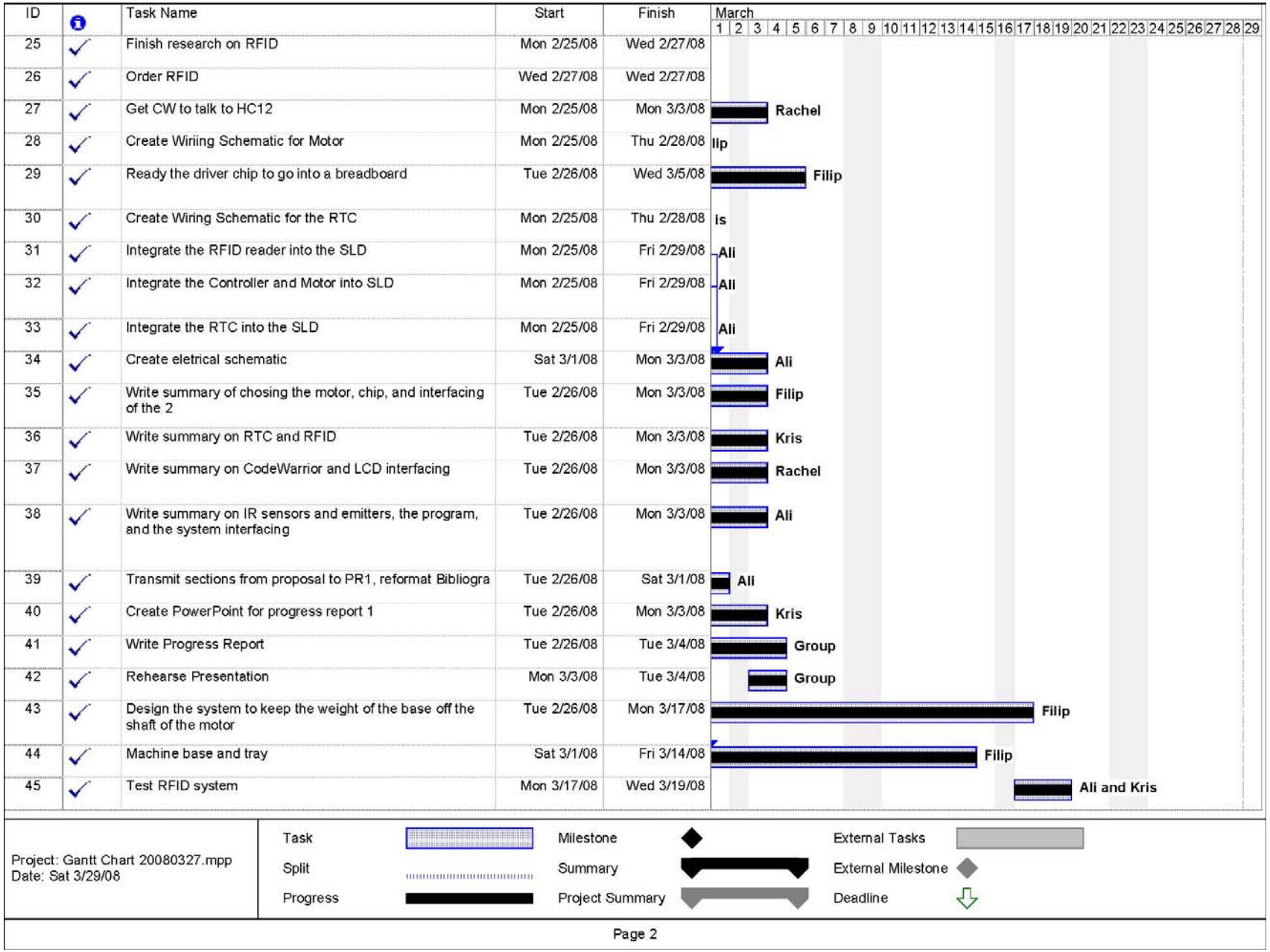


Figure 26: Gantt Chart for the Past Month Part 1

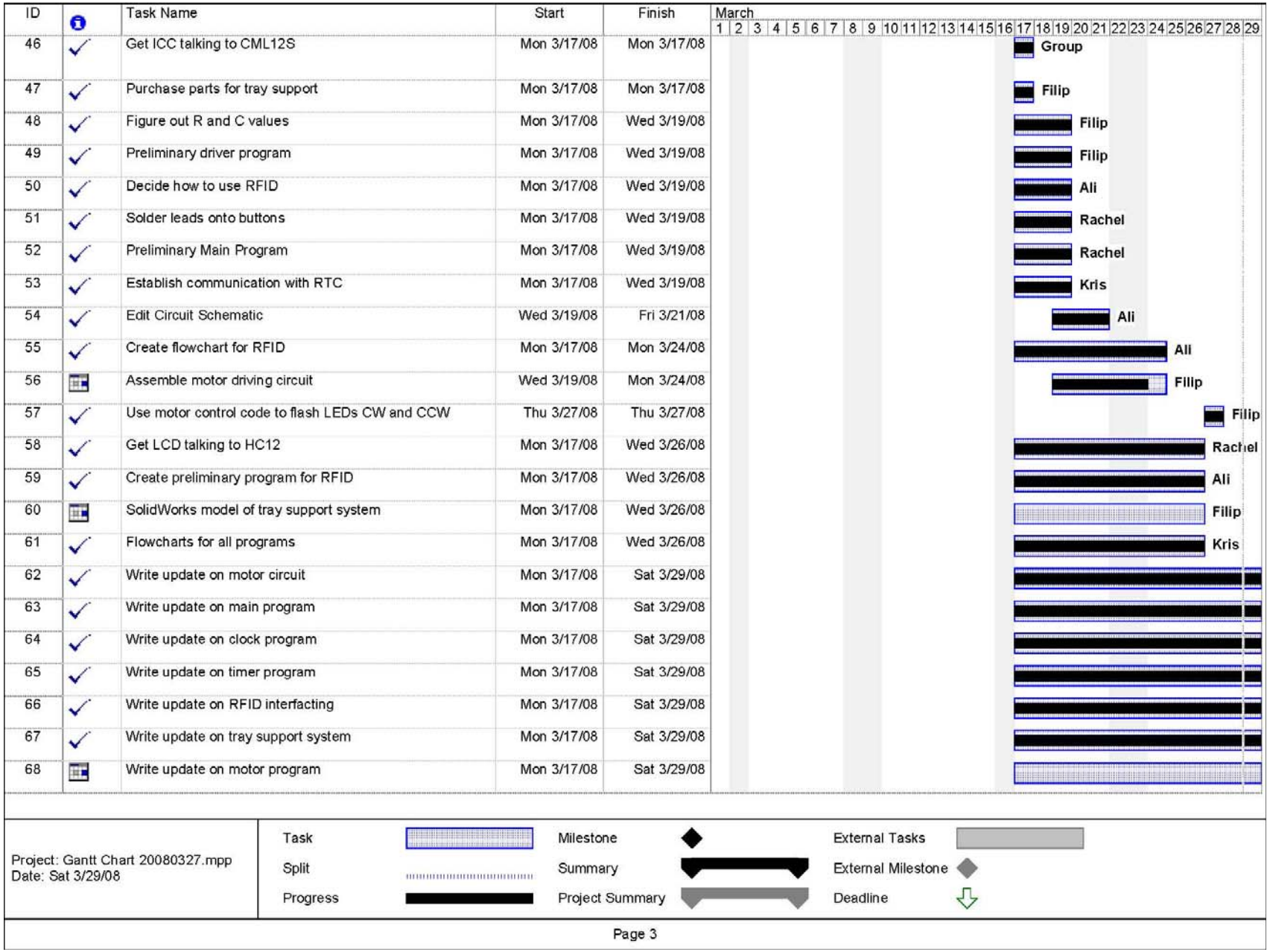


Figure 27: Gantt Chart for the Past Month Part 2

Project References:

[1] Heil, Rachel, et al. The Smart Pet Feeder: Progress Report 1. 05 Mar. 2008.

[2] "RFID Reader Module (#28140)." Parallax, Inc. Rocklin: 2005.

[3] "DS1286 Watchdog Timekeeper." Jameco.com. 03 Mar. 2008. <http://www.jameco.com/webapp/wcs/stores/servlet/ProductDisplay?langId=-1&storeId=10001&catalogId=10001&productId=133444>

[4] "Hexa to Binary and Decimal converter / convertor." Easy Calculation.com. 29 Mar. 2008. <http://www.easycalculation.com/hex-converter.php>

[5] "1.8° Size 17 Super Torque Motor." Lin Engineering. Santa Clara.

[6] "SLA7024M, SLA7026M, and SMA7029M High-Current PWM, Unipolar Stepper." Allegro. Worcester: 1994.

[7] "CML-9S12DP512." Axman.com. 1 Mar. 2008.

[8] Grant, Matthew . "Quick Start for Beginners to Drive a Stepper Motor". Freescale.com. March 30, 2007. March 17, 2007. <http://www.freescale.com/files/microcontrollers/doc/app_note/AN2974.pdf?fsrch=1>

[9] "CML-9S12DP256." Axiom Manufacturing. Garland: 2004.

[10] "MC9S12DP512 Device Guide V01.25." Freescale Semiconductor, Inc. Chandler: 2005

[11] "DS1286 Watchdog Timekeeper." Dallas Semiconductor.

[12] "Full size Toggle Switch." Jameco.com. 03 Mar. 2008. <http://www.jameco.com/webapp/wcs/stores/servlet/ProductDisplay?langId=-1&storeId=10001&catalogId=10001&pa=76232&productId=76232>

[13] "Open drain." Wikipedia.com. 16 Jan. 2008. 29 Feb. 2008. <http://en.wikipedia.org/wiki/Open_drain>

[14] "Staywell Infra-Red Cat Door." Moorepet-petdoors.Com. 28 Feb. 2008 <http://www.moorepet-petdoors.com/PhotoGallery.asp?ProductCode=500US>

[15] “Staywell Infra-Red Collar Key.” Petco.Com. 28 Feb. 2008 <http://www.petco.com/shop/product.aspx?sku=968170&cm_ven=tag&cm_cat=34&cm_pla=968170&cm_ite=968170>

[16] "Radio-Frequency Identification." Wikipedia. 2 Mar. 2008. Wikimedia Foundation, Inc. 20 Feb. 2008 <http://en.wikipedia.org/wiki/RFID>.

[17] Lewan, Todd. "Chip Implants Linked to Animal Tumors." Washingtonpost.Com. 8 Sept. 2007. The Associated Press. 2 Mar. 2008 <http://www.washingtonpost.com/wp-dyn/content/article/2007/09/08/AR2007090800997_pf.html>.

[18] Cheung, Humphrey. "American Medical Association Wants Implantable RFID Chips." TG Daily. 27 June 2007. Tigervision Media. 28 Feb. 2008 <http://www.tgdaily.com/content/view/32663/113/>.

[19] Greene, Thomas C. "Feds Approve Human RFID Implants." The Register. 14 Oct. 2004. 2 Mar. 2008 <http://www.theregister.co.uk/2004/10/14/human_rfid_implants/>.

[20] “PetSafe Electronic SmartDoor Pet Door.” PetFrenzy.com. 3 Mar. 2008.

[21] “Sharp PT481/PT481F/PT483F1 Narrow Acceptance High Sensitivity Phototransistor.” Sharp Electronics.

[22] “PT501/PT510 TO-18 Type Narrow Acceptance High Sensitivity Phototransistor.” Sharp Electronics.

[23] “PT4800/PT4800F/PT4810/PT4810F/PT4850F Thin Type Phototransistor.” Sharp Electronics.

[24] “QSC112, QSC113, QSC114 Plastic Silicon Infrared Phototransistor.” Fairchild Semiconductor. 2005.

[25] “The Overweight Pet.” ThePetCenter.Com. 25 Jan. 2008 <http://www.thepetcenter.com/imtop/overweight.html>.

[26] “Petmate Café Feeder.” Amazon.com. 25 Jan. 2008 <http://www.amazon.com/gp/product/B0002DI2XC/ref=s9_asin_image_1?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=center-2&pf_rd_r=0WDC23RF23M58AJ1CCJG&pf_rd_t=101&pf_rd_p=278240301&pf_rd_i=507846>

[27] "Petmate Le Bistro Electronic Portion-Control Automatic Pet Feeder." Amazon.com. 25 Jan. 2008 <http://www.amazon.com/gp/product/B000BVWVUA/ref=s9_asin_image_2?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=center-2&pf_rd_r=0WDC23RF23M58AJ1CCJG&pf_rd_t=101&pf_rd_p=278240301&pf_rd_i=507846>.

[28] "Petmate Le Bistro Electronic Portion-Control Automatic Pet Feeder Customer Reviews."Amazon.com. 25 Jan. 2008 <http://www.amazon.com/review/product/B000BVWVUA/ref=dp_db_cm_cr_acr_txt?%5Fencoding=UTF8&showViewpoints=1>.

[29] "Perfect Petfeeder Lux Model." Pillar Pet Products, Inc. 25 Jan. 2008 <http://www.perfectpetfeeder.com/default.html>.

[30] "Pet Product Review: Perfect Petfeeder." Itchmo: News For Dogs & Cats. 25 Jan. 2008 <http://www.itchmo.com/pet-product-review-perfect-petfeeder-3428>.

[31] "ERGO 8 Day Feeder." Pet Street Mall. 25 Jan. 2008 <http://www.petstreetmall.com/Automatic-8-Day-Feeder/5052/1896/>.

[32] "How to Keep Your Dog from Eating Your Cat's Food." wikiHow.com. 25 Jan. 2008 <http://www.wikihow.com/Keep-Your-Dog-from-Eating-Your-Cat's-Food>.

[33] "Industry Statistics & Trends". American Pet Products Manufacturers Association, Inc. 25 Jan. 2008 <http://www.appma.org/press_industrytrends.asp>.

[34] "Do You Like Pets Better Than People?". CBS News. 25 Jan. 2008 <http://www.cbsnews.com/stories/2007/09/05/opinion/garver/main3234187.shtml>.

[35] "The Pet Economy". Business Week. 25 Jan. 2008 <http://www.businessweek.com/magazine/content/07_32/b4045001.htm>.

[36] "It's a Pet Economy". Sacramento Business Journal. 25 Jan. 2008 <http://www.bizjournals.com/sacramento/stories/2007/08/20/story3.html>.

[37] “The Overweight Pet”. The Pet Center. 25 Jan. 2008 <http://www.thepetcenter.com/imtop/overweight.html>.

[38] “Dog Owner’s Guide: Obesity”. CanisMajor.com. 25 Jan. 2008 <http://www.canismajor.com/dog/obese.html>.

Appendices:

Appendix A: Preliminary C Language RFID Program

```
/*
 *  RFID program for the Automated Smart Pet Feeder
 *  This program will determine if the RFID reader is receiving a signal from
 *  an RFID tag.

 *  Title: RFID.c
 *  By: Alexis Rodriguez-Carlson
 *  Date: March 26, 2008


   NOTES:
   * The RFID reader outputs a HIGH (1) signal when it is not receiving a
         signal from a tag.  If it is receiving a signal from a tag, it will output
         a unique series of numbers to Port P (PTP) pin 2.  In our case we
         don't care about the identifying sequence, only if there is a tag there
         at all.  So, we will record the each bit as a seperate variable (A-R) and
         then AND those variables together in another variable called "TAG".  The
         zeros in the sequence will result in a zero value for TAG if the forbidden
         pet has approached.  If the forbidden pet is present, the program will
         return a value of PET=1 to be used in other programs.
*/

/**header files to be included**/
#include <mc9s12dp512.h>
#include <stdio.h>


/*define variables*/
int A,B,C,E,F,G,H,J,K,L,N,O,P,Q,S,T,U,V;                /*sample values from RFID
                                                                reader.  Letters D, I,
M, and R
                                                                are missing because
the program
                                                                wouldn't compile
with them for
                                                                reasons I do not
understand.*/

int TAG;                                                /*Value equal to ANDing all
                                                                of the sample
values*/
int PET;                                                /*Variable which is equal to
                                                                1 if the tag is
present*/


rfid(void)
{

DDRP=00;                                                /*this line configures Port P
```

```
                                                                as an input*/

/*This section sets the value of each variable equal to the value in PTP ANDed
by 00100000.  This mask makes it possible to use only the value at PTP2 to set
the variable values.*/

A = PTP&&00100000;
B = PTP&&00100000;
C = PTP&&00100000;
E = PTP&&00100000;
F = PTP&&00100000;
G = PTP&&00100000;
H = PTP&&00100000;
J = PTP&&00100000;
K = PTP&&00100000;
L = PTP&&00100000;
N = PTP&&00100000;
O = PTP&&00100000;
P = PTP&&00100000;
Q = PTP&&00100000;
S = PTP&&00100000;
T = PTP&&00100000;
U = PTP&&00100000;
V = PTP&&00100000;

TAG = A&&B&&C&&E&&F&&G&&H&&J&&K&&L&&N&&O&&P&&S&&T&&U&&V; /*ANDing
all of the

variables together will
                                                                        result
in a value of 0
                                                                        if
there is a tag

present*/


if (TAG == 0)     /*The RFID reader is detecting a tag, the forbidden pet is
                            near*/
{
 PET = 1;
 }
else                        /*The RFID reader is not detecting a tag, the coast is
                            clear!!!*/
{
 PET = 0;
 }

return(PET);}
```

Appendix B: Preliminary C Language Main Program

```
/*
 * Main program for the Automated Smart Pet Feeder
 * This program will determine if the override switch is on or off and
 * determine which subroutine to call. The operation of the entire
 * feeder is determined by this program.

 * Title: main.c
 * By: Rachel Heil
 * Date: March 11, 2008


   NOTES:
   * Override Switch:
     Inputs a 5 Volt signal to Port P1 (PP1) at pin 42 indicating it
     is in the on position, set to high (1)
*/

/**header files to be included**/
#include <mc9s12dp512.h>
#include <stdio.h>

/**Set ports necessary to Inputs or Outputs**/
  DDRP_DDRP1 = 0x00; // Port P set to inputs (0 in each bit)
  DDRK_DDRK1 = 0xFF; // PK0:PK3 - Port K (outputs)(1 in each bit)
  DDRT_DDRT1 = 0xFF; // PTT0:PTT5 - Port T (outputs)(1 in each bit)
  DDRS_DDRS1 = 0xFF; // PS0, PS2, PS4, PS6 - Port S (outputs)(1 in each bit)

/**Assign port addresses**/
   DDRP_PTP1 = 0x0025A&&01000000;     //address of Port P1



/***Reference Addresses***/
/**Addresses of Direction Registers**/
/* DDRP = 0x025A;                    //address of DDRP
   DDRK = 0x0033;                    //address of DDRK
   DDRT = 0x0242;                    //address of DDRT
   DDRS = 0x024A;                    //address of DDRS
*/
/**Addresses of ports**/
/*DDRK_PORTK = 0x0032;                  //address of Port K
  DDRT_PTT = 0x0240;                   //address of Port T
  DDRS_PTS = 0x0248;                   //address of Port S
 */



/**Define all functions and subroutines**/
void clockset(void);   //defines clock set function
void detect(void);     //define pet detect function
```

```
void feeding(void);     //define feeding time function

/**Define all variables necessary**/



/**Body of main program**/

int main(void)                    // beginning the main program
{
 if (DDRP_PTP1=1)      //Port P1 is set to high, override switch is on
   {
          clockset();     //calls the clockset subroutine
   }
 else              //Port P1 is not set to high and override switch is off
   {
          detect();              //calls the petdetect subroutine
   }
 return(0);           //signals end of program
}
```

Appendix C: C Language LCD Programs

```
/* ============================================================================

   lcdtest.c  - Test program for LCD.C

   Version:       1.0
   Author:        Dusty Nidey, Axiom Manufacturing (www.axman.com)
   Compiler:      GNU for 68HC11 & 68HC12 (www.gnu-m68hc11.org)

   This is freeware - use as you like

============================================================================
----------------------------------------------------------------------------
*/

#include "..\ports_d256.h"
#include "lcd.h"


main(){
   char keyval;

   LCDInit();  // initialize LCD

   LCDputs("Hello  1234567890_AB");    // line 1
   LCDputs("Line 3 1234567890_AB");    // line 3
   LCDputs("Line 2 1234567890_AB");    // line 2
   LCDputs("Line 4 1234567890_AB");    // line 4
}


// -------------
unsigned char LCDBuf;   // holds data and status bits sent to LCD
unsigned char LCDStat;  // holds LCD status

#define LCD_DELAYTIME   0x200   // adjust this value Lower for quicker LCD
//performance, Higher if you're seeing garbage on the display

// simple delay loop, waits the specified number of counts
void LCD_delayu(unsigned int ucount){
   while(ucount > 0){
      --ucount;
      // look at your compiler output and count the number of cycles used.
      // add more of these if needed to fine tune the exact delay you want
      //       asm("nop");

   }
}
// simple delay loop, LCD_delayu() * LCD_DELAYTIME
void LCD_delaym(unsigned int mcount){
   while(mcount > 0){
```

```
      --mcount;
      LCD_delayu(LCD_DELAYTIME);
   }
}

// Simple Serial Driver (SPI) send byte
// sends data byte in global LCDBuf
// return received back value in global LCDStat
void LCDSend(){
   LCD_delaym(1);
   LCDStat = SPI0SR;   // clear status of SPI by reading it
   SP0DR = LCDBuf; // send byte

   do{ // wait for status flag to go high
      LCDStat = SPI0SR;
   }while(LCDStat < 0x80);

   LCDStat = SP0DR;    // receive value back
}

// writes 4 bit data to lcd port
void lcd_wr_4(char LCDdata){
   // merge lower 4 bits of LCDdata with upper 4 bits of LCDBuf (control bits)
   LCDBuf &= 0xF0;
   LCDBuf |= LCDdata;

   LCDBuf &= ~EN;  // enable low
   LCDSend();  // send the data
   LCDBuf |= EN;  // enable high
   LCDSend();  // send the data
   LCDBuf &= ~EN;  // enable low
   LCDSend();  // send the data
}

// same as above but with delay
void lcd_wr_4d(char LCDdata){
   lcd_wr_4(LCDdata);
   LCD_delaym(50);
}
// Lcd Write 8 bit Data , upper 4 bits first, then lower
void LCD_Write(unsigned char lcdval){
   lcd_wr_4(lcdval >> 4);      // send upper 4 bits
   lcd_wr_4(lcdval & 0x0F);    // send lower 4 bits
}
// write a COMMAND byte to the LCD
void lcd_cmd(unsigned char cmdval){
   LCDBuf &= ~RS; // clear RS to select LCD Command mode
   LCD_Write(cmdval);
   LCD_delaym(10);
}
```

```
/*---------------------------------------------------------------------------
LCDputch
--------
 Send a character to the LCD for display.

 INPUT:     datval      character to display
---------------------------------------------------------------------------*/
void LCDputch(unsigned char datval){
   LCDBuf |= RS; // set RS to select LCD Data mode
   LCD_Write(datval); // write a DATA byte to the LCD
}

/*---------------------------------------------------------------------------
LCDputs
-------
 Send a string of characters to the LCD.  The string must end in a 0x00

 INPUT:                  sptr                pointer to the string
---------------------------------------------------------------------------*/
void LCDputs(char *sptr){
          while(*sptr){
       LCDputch(*sptr);
                  ++sptr;
          }
}

/*---------------------------------------------------------------------------
LCDInit
-------
 Initialize LCD port

 ---------------------------------------------------------------------------*/
void LCDInit(){

   //  Turn on Spi
   SPI0CR1 = 0x52;
   SPI0CR2 = 0x10; // enable /SS
   SPI0BR = 0x00;  // set up Spi baud clock rate
   LCDBuf = (WR + EN); // set WR and EN bits
   LCDSend();  // send status to LCD

   // Initialize LCD
   LCDBuf &= ~(RS + EN); // clear RS and EN bits (select lcd Command)
   LCDSend();  // send status to LCD

   // delay's are used because this lcd interface does not provide status
   LCD_delaym(50);

   // set to 4 bit wide mode
```

```
    lcd_wr_4d(3);   // send 3
    lcd_wr_4d(3);   // send 3
    lcd_wr_4d(3);   // send 3
    lcd_wr_4d(2);   // send 2

    lcd_cmd(0x2c);  // 2x40 display
    lcd_cmd(0x06);  // display and cursor on
    lcd_cmd(0x0e);  // shift cursor right
    lcd_cmd(0x01);  // clear display and home cursor
    lcd_cmd(0x80);

    LCDBuf = 0; // Reset Lcd states to rest
    LCDSend();  // send status to LCD
}
```

Appendix D: Preliminary C Language Feeding Time Program

```
/* Program Name:  Feeding Time
   Written By:  Alexis Rodriguez-Carlson
   Revision Date:  March 20, 2008
   Purpose:  This program determines if the MCU is receiving an alarm signal
                        from the RTC.  If it is, it rotates the food tray to reveal the
                        next meal, if it is not, it calls the Pet Detect program.*/

/**header files to be included**/
#include <mc9s12dp512.h>
#include <stdio.h>

RFID(void);                 /*sub-program which checks to see if the RFID
                                        is receiving a signal from the tag*/
ccw(void);          /*sub-program which rotates the tray 1 bowl
                                        counter clockwise*/
cw(void);           /*sub-program which rotates the tray 1 bowl
                                        clockwise*/
timer(void);                /*sub-program which used the RTC to count thirty sec*/
clockset(void);             /*sub-program which allows user to set the clock
                                        and meal times*/
detect(int BOWL);

int ALARM;                              /*variable which will be equal to 0 is the RTC is
                                        NOT sending an alarm*/
int BOWL;                               /*variable which indicates which dish is to be
                                        revealed*/
int COUNTER;                            /*variable used to count how many times the motor
                                        rotation program will run*/
int PET;                                /*variable that will equal 1 if forbidden pet is
                                        detected*/

/*program code is below*/

feeding(BOWL)     /*this program must take the variable BOWL as it
                                        exists from the other sub-programs, change it, and
                                        then return the new value so that other
                                        sub-programs can use*/


{
 if (PTP && 01000000 == 00000000)  /*check to see if the manual override switch
                                                is on.  If PTP1 is equal to 1, then the
                                                IF statement is valid*/
 {
  ALARM = PTP && 00001000;        /*Sets ALARM to be equal to 00001000 if PTP4
                                                is equal to 1, or to 00000000 if PTP4 is
                                                equal to 0*/

  if (ALARM==00001000)                  /*check to see if the RTC is sending an
                                                alarm*/
```

```
 {
  BOWL = BOWL+1;                                  /*sets the number of the next bowl to be
                                                        revealed*/
  RFID();                                         /*call RFID program to see if the forbidden
                                                        pet is near*/

  if (PET==1)                                     /*check to see if the RFID reader is
                                                        receiving a signal*/
  {
   COUNTER=BOWL-1;                                /*set the variable COUNTER to be equal the
                                                        number of the bowl which is currently
                                                        exposed*/
        while (COUNTER > 0)                       /*this loop rotates the tray to reveal the
                                                        blank spot*/
   {
    ccw(void);                    /*calls the counter clockwise rotation
                                                        program*/
    COUNTER = COUNTER-1;                          /*decrements COUNTER*/
    }
   while (PET==1)                                 /*this loop checks to see if the RFID reader
                                                        is receiving a every thirty seconds*/
   {
    RFID();
         timer(void);
   }
   while (COUNTER < BOWL-1)       /*rotates the tray back to the bowl before
                                                        the revealed bowl*/
   {
    cw(void);                     /*calls the clockwise rotation program*/
    COUNTER = COUNTER +1;         /*increments COUNTER*/
   }
  }
  cw(void);                /*rotates tray to reveal new bowl*/
  if (BOWL==7)                                    /*resets BOWL to 0 if it has reached 7
                                                        (since there are only 6 bowls*/
    {
         BOWL=0;
         }
 }
 detect(BOWL);                        /*call the detect program*/
}
clockset(void);                                   /*call the clockset program*/
return(BOWL);
}
```

Appendix E: Preliminary C Language Pet Detect Program

```
/* Program Name:  Pet Detect
   Written By:  Alexis Rodriguez-Carlson
   Revision Date:  March 20, 2008
   Purpose:  This program determines if the MCU is receiving a signal from the
                        RFID reader.  If it is, it rotates the food tray to the blank spot,
                        if it is not, it calls the Feeding Time program.*/



/**header files to be included**/
#include <mc9s12dp512.h>
#include <stdio.h>

/*define sub-programs*/
RFID(void);                     /*sub-program which checks to see if the RFID
                                                        is receiving a signal from the tag*/
ccw(void);              /*sub-program which rotates the tray 1 bowl
                                                        counter clockwise*/
cw(void);               /*sub-program which rotates the tray 1 bowl
                                                        clockwise*/
timer(void);            /*sub-program which used the RTC to count thirty
                                                        sec*/
clockset(void);                 /*sub-program which allows user to set the clock
                                                        and meal times*/
feeding(int BOWL); /*sub-program which checks to see if the the RTC
                                                        is sending an alarm signal to indicate a meal
                                                        time*/

/*define variables*/

int ALARM;                                      /*variable which will be equal to 0 is the RTC is
                                                        NOT sending an alarm*/
int BOWL;                                       /*variable which indicates which dish is to be
                                                        revealed*/
int COUNTER;                                    /*variable used to count how many times the motor
                                                        rotation program will run*/

int PET;                                        /*variable which equals 1 if forbidden pet is
                                                        */
int OVERRIDE;                                   /*variable which equals 1 if override is on*/


void detect(BOWL)
{
  OVERRIDE = PTP&&01000000;
  if (OVERRIDE == 00000000)  /*check to see if the manual override switch
                                                        is on.  If PTP1 is equal to 0, then the
                                                                IF statement is valid*/
  {
   RFID(void);
```

```
  if (PET==1)                    /*check to see if the RFID reader is
                                                receiving a signal*/
  {
   COUNTER=BOWL;         /*set the variable COUNTER to be equal the
                                                number of the bowl which is exposed*/


   while (COUNTER > 0)      /*rotates the tray to reveal the blank spot*/
   {
    ccw(void);           /*calls the counter clockwise rotation
                                                program*/
    COUNTER = COUNTER-1;       /*decrements COUNTER*/
    }
    while (PET==1)                    /*this loop checks to see if the RFID reader
                                                is receiving a every thirty seconds*/
   {
    RFID();
          timer(void);
   }
   while (COUNTER < BOWL)       /*rotates the tray back to the revealed
                                                bowl*/
   {
    cw(void);            /*calls the clockwise rotation program*/
    COUNTER = COUNTER + 1; /*increments COUNTER*/
   }
  }
  feeding(BOWL);
 }
  clockset(void);

}
```

Appendix F: Preliminary C Language Motor Rotation Program

```
#define NUM_OF_STATES 8 //There are 8 different states in this particular example.
#define DELAY_MAX 5000  //The maximum # of counts used to create a time delay.
#include <STDIO.h>
#include <hc11.h>

void main(void)
{
/*******************CREATE VARIABLES*******************/
int i; //Used in a for loop

//This array actually contains the state values that will be placed on Port B.
//State #0 corresponds to a value of 0x06, state #1 corresponds to a value of 0x02, etc.

char state_array[NUM_OF_STATES] = {0x06, 0x02, 0x0A, 0x08, 0x09, 0x01, 0x05, 0x04};
                                  //0110, 0010, 1010, 1000, 1001, 0001, 0101, 0100)
int steps_to_move;                //The # of rotational steps the motor will make.
char next_state;                  //Used to select the next state to put in register B.


/*******************SET UP PORT B*******************/
//DDRK = 0xFF; //Writing 0xFF to DDRK sets all bits of Port K to act as output.

PORTB = 0;     //Init Port B by writing a value of zero to Port B.
/*****************************************************/


steps_to_move = 1;              //Set the # of steps to move. An arbitrary positive # can be used.

next_state = 0;                       //Init next_state to state 0. next_state can start from any state
                                      //within the range of possible states in this example, 0-7.

PORTB = state_array[next_state];//Init Port B to the starting state. In this example,
                                                        //since only 4 pins are needed to control the motor,
only
                                                        //the lower nibble of Port B is being used. This line
                                                        //selects state 0 and places the corresponding value
                                                        //(0x06) in the lower nibble of Port B.

for(i = 0; i < DELAY_MAX; i++)
{
 //Wait here for a while.
}
while (steps_to_move > 0)
{
 if (next_state > (NUM_OF_STATES - 1))                  //If next_state is greater than the highest
                                                        //available state, 7, then cycle back to 0
 {
   next_state = 0;
 }
```

```
 PORTB = state_array[next_state];                    //Place new value in Port B. Rotation may be
observed
 for(i = 0; i < DELAY_MAX; i++)
 {
                                                     //Wait here for a while.
 }
 next_state++;                                       //Increment next_state. Cycling though the states
                                                       causes
                                                     //rotation in one direction. Decrementing states
                                                       causes
                                                     //opposite rotation.

 steps_to_move--;                                    //Subtract 1 from the total # of steps remaining to be
                                                     //moved.
}

//The following code rotates the motor back in the opposite direction.

steps_to_move = 100;
while (steps_to_move > 0)
{
 if (next_state < 0)
 {
  next_state = (NUM_OF_STATES - 1);
 }
 PORTB = state_array[next_state];
 for(i = 0; i < DELAY_MAX; i++)
 {
  //Wait here for a while.
 next_state--; }
 steps_to_move--;
}
}          //End of Main
```

Appendix G: Microcontroller Block Diagram and Datasheet

CML12S-DP256

Development Board for Motorola MC9S12DP256

Axiom Manufacturing • 2813 Industrial Lane • Garland, TX 75041
Email: Sales@axman.com Web: http://www.axman.com

CONTENTS

GETTING STARTED

The Axiom CML12S-DP256 single board computer is a fully assembled, fully functional development system for the Motorola MC9S12DP256 microcontroller. Provided with wall plug power supply and serial cable. Support software for this development board is provided for Windows 95/98/NT/2000/XP operating systems.

This development board applies option selection jumpers. Terminology for application of the option jumpers is as follows:

> Jumper on, in, or installed = jumper is a plastic shunt that fits across 2 pins and the shunt is installed so that the 2 pins are connected with the shunt.
>
> Jumper off, out, or idle = jumper or shunt is installed so that only 1 pin holds the shunt, no 2 pins are connected, or jumper is removed. It is recommended that the jumpers be idled by installing on 1 pin so they will not be lost.

Development board users should also be familiar with the hardware and software operation of the target HCS12 device, refer to the Motorola User Guide for the device and the CPU12 Reference Manual for details. The development board purpose is to assist the user in quickly developing an application with a known working environment or to provide an evaluation platform for the target HCS12. Users should be familiar with memory mapping, memory types, and embedded software design for the fastest successful application development.

Application development maybe performed by applying the embedded MON12 (**default**) or NOICE firmware monitors, or by applying a BDM cable with supporting host software. The MON12 monitor provides an effective debug method for assembly level software, but has limitations in C code developments. For C/C++ code development it is recommended that source code or symbolic debug capability be provided in the debugging environment. The NOICE monitor or BDM interface with supporting software tools should be applied for C/C++ code development so the host PC can provide the symbolic support needed. User should verify the NOICE or BDM development environment supports the C compiler to be applied, not all development environments support all compilers.

The MON12 and NOICE monitors are provided in the development board HCS12 internal flash memory and apply some HCS12 resources for operation. See the respective chapter for each monitor for details on operation and resources applied. User should note both monitors apply operation of the HCS12 expanded wide mode data and address bus on HCS12 I/O ports A, B, E, and K for access to the external Ram. The external ram provides a development memory where code to be debugged can be loaded or modified quickly and software breakpoints applied. After the application is tested, the code can be relocated to the internal flash memory space of the HCS12 and programmed into the flash memory for dedicated operation.

User applications developed by applying MON12 or NOICE monitors can be modified and relocated for operation as a stand-alone application. By applying the MON12 Autostart feature, the user application will operate from Reset or Power on conditions to provide a dedicated operation of the application. See the Autostart section in this manual for more information.

Follow the steps in this section to get started quickly and verify everything is working correctly.

Installing the Software

1. Insert the Axiom 68HC12 support CD in your PC. If the setup program does not start, run the file called "SETUP.EXE" on the disk.
2. Follow the instructions on screen to install the support software onto your PC. You should at minimum install the AxIDE for Windows software.
3. The programming utility "AxIDE" requires you to specify your board. You should select "CML12SDP256" version of your development board.

Board Startup

Follow these steps to connect and power on the board for the default Monitor operation. This assumes you're using the provided AxIDE utility (installed in the previous section) or a similar communications terminal program on your PC. If you're using a different terminal program than the one provided, set it's parameters to 9600 baud, N,8,1.

1. Set the CML12Sxxx board Option jumpers to default positions:

   MEM-EN = IN, ECS = IN, JP1 = IN, NOAUTO (SPARE) = IN

   MODC = Out, JP2 = do not care, see COM Ports.

2. Connect one end of the supplied 9-pin serial cable to an available serial COM port on your PC. Connect the other end of the cable to the P-COM port on the CML12Sxxx board.
3. Apply power to the board by plugging in the power adapter that came with the system.
4. If everything is working properly, you should see a message to "**PRESS KEY TO START MONITOR...**" in your terminal window. Press the ENTER key and you should see:

```
Axiom MON12 - HC12 Monitor / Debugger V256.x
Type "Help" for commands...

> _
```

5. Your board is now ready to use! If you do not see this message prompt, or if the text is garbage, see the **TROUBLESHOOTING** section at the end of this manual.

Support Software

There are many programs and documents on the included HC12 support CD you can use with the CML12Sxxx board. You should install what you want from the main menu then browse the disk and copy what you like to your hard drive.

At minimum, you should install the AxIDE program. This provides the flash programming utility and communication with the board via the COM port and the supplied serial cable. This program includes a simple terminal for interfacing with other programs running on the CML12Sxxx and information from your own programs that send output to the serial port.

Also on the disk are free assemblers AS12 and MCU-EZ, the open source GNU C/C++ compiler tools for HC11/12, example source code, and other useful software. The introductory tutorial in this manual uses the free AS12 assembler integrated into the AxIDE program. This is a simple assembler with limited capability. For a more powerful assembly tool, install the Motorola MCUez program from the CD. This will allow you to use PAGED program memory in your application.

Software Development

Software development on the CML12Sxxx can be performed using either the MON12 monitor installed in internal FLASH of the MCU, a third party debugger (Debug12, NoICE, CodeWarrior, etc.) or a Background Debug Module (BDM) connected to the BDM PORT connector. Any of these tools can be used to assist in creating and debugging your program stored in RAM (see **Memory Map**).

After satisfactory operation running under a debugger, your program can be written to Internal Flash Memory using the included programming utilities. The Mon12 firmware in the MCU flash provides the interrupt vectors in Ram memory and an Autostart feature to launch your application. Your program may then run automatically whenever the board is powered on or RESET is applied.

TUTORIAL

This section was written to help you get started developing software with the CML12SXXX board. Be sure to read the rest of this manual as well as the documentation on the disk if you need further information.

The following sections take you through the complete development cycle of a simple "hello world" program, which sends the string "Hello World" to the serial port.

Creating Source Code

You can write source code for the CML12SXXX board using any language that compiles to Motorola 68HC12 instructions. Included on the software disk is a free Assembler, AS12.

You can write your source code using any ASCII text editor. You can use the free EDIT, WordPad, or Notepad programs that come with your computer. Note that the source file must be simple ASCII text without any document formatting added. Once your source code is written and saved to a file, you can assemble or compile it to a Motorola S-Record (hex) format. This type of output file usually has a .MOT, .HEX or .S19 file extension and is in a format that can be read by the programming utilities and programmed into the CML12SXXX board.

It is important to understand the development board's use of Memory and Addressing when writing source code so you can locate your code at valid addresses. For example, when in debug mode, you should put your program CODE in External RAM. In assembly language, you locate the code with ORG statements in your source code. Any lines following an ORG statement will begin at that ORG location, which is the first number following the word ORG, for example: `ORG $4000`. You must start your DATA (or variables) in a RAM location unused by your program, for example: `ORG $1000`.

In "debug mode" you'll be using a debugger utility (Mon12, NoICE, etc) which will handle initialization, interrupts vectors (reset, timers, etc), and the STACK. When finished debugging, you must add code to your application to handle the initialization of the CPU, STACK and possibly the Interrupt vectors. Some initialization is required to set the bus frequency, bus mode, internal EEprom and Flash memory programming clock rates, and others, see the CML-INIT.ASM file for a sample. Set the stack at the top of your available internal RAM below the Ram interrupt vector table, for example $3F80, in assembly this would be `LDS #$3F80`. Also install the RESET vector address in the Auto Start area, see the chapter in this manual.

If you are applying a software development tool that also provides a BDM cable interface to the board, the monitor installed in the flash is not required. The BDM software tools may have the capability to erase and program the flash memory. If this is the case, you may develop code in the external ram memory or internal flash without applying the monitor resources. The MON12 S record is provided on the support CD to program into the flash if desired. The BDM will allow locating programs in memory and applying resources reserved for the monitors.

A look at the example programs on the disk can make all of this clearer. If you're using a compiler instead of an assembler, consult the compiler documentation for methods used to locate (MAP) your code, data and stack.

Assembling source code

An example program called "HELLO.ASM" is provided under the **\EXAMPLES\CML12** directory of the CD and if you installed AxIDE, under that programs **\EXAMPLE** directory. You must use the example for the MCU type installed on the CML12Sxxx board. For example use the CML12S-DP256 example on the DP256 version board.

You can assemble source code by using the AxIDE "BUILD" button or command line tools under a DOS prompt by typing:

```
AS12 HELLO.ASM -L HELLO
```

Most compilers and assemblers allow many command line options so using a MAKE utility or batch file is recommended if you use this method. Run AS12 without any arguments to see all the options, or see the AS12.TXT file on the disk.

The utility software, AxIDE, provided with this board contains a simple interface to this assembler. Use it by selecting "Build" from its menu. This will prompt you for the file to be assembled. **NOTE**: You must select your board from the pull down menu first, or it may not build correctly.

DO NOT use long path or file names (> 8 characters). The free assembler is an older DOS based tool that does not recognize them.

If there are no fatal errors in your source code, 2 output files will be created:

`HELLO.S19`	a Motorola S-Record file that can be loaded or programmed into memory
`HELLO.LST`	a common listing file which provides physical address information with resulting opcode and operand information. Warnings and error messages are provided with a summary at the end of the file.

The listing file is especially helpful to look at when debugging your program. If your program has errors, they will be displayed in the listing or fatal errors will prevent output from being generated. The end of the listing file generally provides a count of errors or warnings in the file.

If you prefer a windows integrated programming environment, try the Motorola MCU-EZ tools. Refer to the MCU-EZ documentation on the disk for more information.

Also, a port for the free GNU C compiler and tools for the HC12 is available on the CD under \Shareware and also online at www.gnu-m68hc11.org. Note that this version does not support HC12 Paging operation, check the web site for updates.

Running your application

After creating a Motorola S-Record file you can "upload" it to the development board for a test run. The provided example "HELLO.ASM" was created to run from external RAM so you can use the MON12 Monitor to test it without programming it into Flash.

If you haven't done so already, verify that the CML12Sxxx board is connected and operating properly by following the steps under "GETTING STARTED" until you see the Mon12 prompt, then follow these steps to run your program:

1. Press and release the RESET button on the CML12Sxxx board. You should see the PRESS ANY KEY message. Hit the return key ↵ to get the monitor prompt.

2. Type `LOAD` ↵
   This will prepare Mon12 to receive a program.

3. Select Upload and when prompted for a file name select your assembled program file in s-record format that was created in the previous section called: `HELLO.S19`
   Your program will be sent to the board through the serial port.

4. When finished loading you will see a done message and the > prompt again. Type `GO 4000` ↵
   This tells MON12 to execute the program at address $4000 hex, which is the start of our test program.

5. If everything is working properly you should see the message "Hello World" echoed back to your terminal screen. Press RESET to return to the monitor.

6. If you do not get this message, see the **TROUBLESHOOTING** section in this manual

You can modify the hello program to display other strings or do anything you want. The procedures for assembling your code, uploading it to the board and executing it remain the same. MON12 has many features such as breakpoints, memory dump and modify and simple program trace (no redirect of the PC is followed). Type HELP at the MON12 prompt for a listing of commands or consult the Mon12 documentation on the disk for more information.

For a more powerful debugger with many advanced features such as source level debugging, you can use the NoICE debugger software. A full-featured demo version is provided on the CD, which you can use to get started. **NOTE**: To use this program instead of MON12 you must set the Autostart, see the NOICE chapter for details.

Programming HCS12 Flash EEprom

After debugging, you can program your application into Flash Memory so it executes automatically when you apply power to the board as follows:

1. Make a backup copy of HELLO.ASM then use a text editor to modify it.

2. Remove the comment ';' character before one of the following lines to initialize the stack pointer which is necessary when running outside of a debugger:

```
LDS     #$3F80    ; initialize stack location...
```

3. Re-Assemble HELLO.ASM as described in the "Assembling Source Code" section.

4. Select **Program** from the AxIDE menu and follow the message prompts. When prompted for a file name, enter the new HELLO.S19 file.

5. Press the RESET button on the board before clicking OK. When prompted to Erase, choose Yes.

6. When finished programming, Reset the board to get the Monitor prompt again. Use the monitor **AUTO** command to set the Autostart Reset vector:

```
>AUTO 4000 ↵

  AutoStart ON, effective address = 4000

>
```

7. Verify AUTO OFF option jumper is not installed. (Spare jumper on revision C boards)

8. RESET or re-apply Power to the board. Your new program should start automatically and the "Hello World" prompt should be displayed in the terminal window.

To return to the MON12 monitor program, install the AUTO OFF option jumper then press RESET. Execute the monitor command NOAUTO to disable the Autostart and allow removal of the Auto Off option jumper for normal operation.

MON12 OPERATION

Mon12 is an embedded monitor / debug utility that allows loading a compiled software program (S record) into Ram memory for testing and debug. The monitor may control the execution of the software by applying the SWI software interrupt service. Other features allow memory and register examination or modification.

Communication with the monitor is provided on the HCS12 SCI0 serial port or COM port on the development board. Default settings are 9600 baud with 8/n/1 bit settings. Flow control is not provided so the host PC communication software should be set to None or Hardware flow control. AxIDE utility software is recommended for use on a windows based host PC.

The monitor relies on resources from the HCS12 target to provide the monitor environment. The resources include 16K bytes of flash memory and 512 bytes of internal ram memory. The user must respect the monitor's memory map when applying the monitor to help debug code. Restricted memory areas:

Monitor Program space: **0xC000 - 0xFFFF Flash or Flash Page $3F.**

Monitor Data space: **0x3E00 - 0x3FFF, Internal Ram.**

Monitor Console: **COM Port and SCI0.**

Monitor Autostart: **0xFEC - 0xFEF, Internal EEprom.**

Monitor application provides for redirection of interrupt vectors through the ram based interrupt table, initialization of SCI0 serial port, initialization of HCS12 flash and EEprom programming clock rates, initialization of 8 MHz E clock from 4Mhz reference crystal, and detection of auto start enabled operation. The HCS12 memory map is fixed under monitor operation.

The monitor provides for interrupt vectors in the monitor data space from 0x3F8A - 0x3FFD. The vectors are in the same order as the default hardware table for the HCS12 located at address 0xFF8A - 0XFFFD (see table). The Reset vector is reserved, user should apply Auto Start for application starting from Reset.

MON12 operation notes:

1. CML12S-DP256 monitor application configures target HCS12 for 8MHz E clock, lower flash block (page $3E) disabled from memory map, and external access clock stretch set to 3 cycles. User can increase clock speed in application by modifying PLL control and setting new baud rate for serial port. Defaults will return whenever monitor is Reset.

2. Mon12 will not trace into interrupts. To trace an interrupt service set a breakpoint in the service routine and then trace.

3. Mon12 trace is limited to expecting the next linear address. Program counter modification, branches, calls, or subroutines will not trace correctly.

4. Monitor start-up procedure:

   A) Determine if Auto Start in enabled and proceed to vector if not a value of $FFFF.

   B) Set Stack, Initialize memory map and SCI0 port and send prompt.

   C) Receive first character from Console port and execute monitor if ASCII text / command, else start utility mode for programming services.

Mon12 Monitor Commands

`AUTO [<Address>]`	Enable Auto start, address is the vector
`NOAUTO`	Disable Auto start
`BF <StartAddress> <EndAddress> [<data>]`	Fill memory with data
`BR [<Address>]`	Set/Display user breakpoints
`BULK`	Erase entire on-chip EEPROM contents
`CALL [<Address>]`	Call user subroutine at <Address>
`G [<Address>]`	Begin/continue execution of user code
`HELP`	Display the Mon12 command summary
`LOAD [P]`	Load S-Records into memory, P = Paged S2
`MD <StartAddress> [<EndAddress>]`	Memory Display Bytes
`MM <Address>`	Modify Memory Bytes (8 bit values)
`MW <Address>`	Modify memory Words (16 bit values)
`MOVE <StartAddress> <EndAddress> <DestAddress>`	Move a block of memory
`RD`	Display all CPU registers
`RM`	Modify CPU Register Contents
`STOPAT <Address>`	Trace until address
`T [<count>]`	Trace <count> instructions

MON12 Interrupt Support

All interrupt services under MON12 are provided through the ram vector table, **see Table 2.** Each location in the table is initialized to a value of $0000 to cause the trap of an unscheduled interrupt. Any nonzero value will allow the interrupt to proceed to the user's service routine that should be located at the provided address value. Interrupt service delay is plus 21 cycles over standard interrupt service.

To use vectors specified in the table, the user must insert the address of the interrupt service routine during software initialization into the ram interrupt table. For an example, for the IRQ vector, the following is performed:

Example: IRQ Service routine label = IRQ_SRV
Ram Vector Table address is defined in table below, IRQ vector definition:
VIRQ EQU $3EF2 ; define ram table vector location

Place IRQ service routine address in the table:
MOVW #IRQ_SRV,VIRQ

This vector initialization should remain after debug when auto start will be applied for launching the user's application. Note that the user interrupt service routines must be located in the $4000 - $7FFF address range for correct operation. See Autostart for more details.

MON12 and NOICE Memory Map

ADDRESS	TYPE MEMORY	MEMORY APPLICATION
$C000 - $FFFF	FLASH	MON12, NOICE, and Utility firmware located in internal flash, Page $3F.
$8000 - $BFFF	External Ram	User Paged Program Memory space, pages $20 - $2E. Note: Pages $30 - $3F reside in the internal flash.
$4000 - $7FFF	External Ram	User Program Memory, emulate fixed page $3E.
$3F8C - $3FFD	Internal Ram	Ram Interrupt Vector Table
$3E00 - $3F8B	Internal Ram	Monitor reserved ram memory. Stacks and variables.
$1000 - $3DFF	Internal Ram	User Internal Ram memory
$0400 - $0FEB	Internal EEprom	User EEprom memory, Monitor reserves $FEC - $FEF for Autostart, user should avoid $FF0 - $FFF memory use.
$0000 - $03FF	HCS12 Registers	Monitor or user access to control registers.

NOICE OPERATION

NOICE is a development software provided by www.NOICEdebugger.com. NOICE provides a development environment that is supported by the NOICE host PC software. This development environment has the capability to provide symbolic debug for C source codes and compilers for a low cost. A fully functional software version is available on the support CD that will operate in demonstration mode. The user should register the software and download the latest version from the above web site to get full support. See the NOICE documentation for details.

The CML12S-DP256 provides the NOICE debug monitor kernel as a subset of the MON12 monitor in reserved flash memory. User may apply the NOICE development system by setting the MON12 Autostart for the $F800 vector, reset the board and launch the NOICE host software on the PC. The NOICE monitor kernel applies the same resources, memory map, and ram interrupt table as the MON12 monitor. NOICE operation notes:

Baud Rate = 19.2K baud 8/n/1

E clock frequency = 24MHz

BDM OPERATION

The CML12S-DP256 board will emulate supported HC12 device internal flash memory in external ram. This feature allows BDM (Background Debug Modules) such as the AX-BDM12 to load and control the execution of code being developed without the necessity of the internal flash memory being programmed many times during the development process. This feature improves updating time and allows the use of may software breakpoints instead of being limited to only 2 hardware breakpoints.

Operation Notes for BDM use:

1) CML12S-DP256 **MODC Option Jumper** should be installed if a BDM is connected to the BDM Port. Default Mode is single-chip so the MODC option installed will force Special Single-chip Mode on Reset.

2) The BDM initialization of the HC12 should set the correct operating MODE (Expanded Wide for memory access). The EME, EMK, LSTRB, RW, ROMEN and Stretch configuration bits should be set for proper external memory access operation. The Axiom support CD contains sample set-up macros for the AX-BDM12.

3) While using the BDM, the user has full control over the memory map and hardware resources of the HCS12. The no resources are required to be reserved for monitor use and the user can apply the actual HCS12 interrupt vector table located at 0xFF8C - 0xFFFF.

AUTOSTART

The MON12 Monitor allows an Autostart operation to launch user applications programmed into the HCS12 internal flash fixed page ($3E) addresses 0x4000 - 0x7FFF from Reset. The Autostart mask and vector are stored in the nonvolatile internal EEprom at addresses $FEC - $FEF. The monitor provides special commands, **AUTO** and **NOAUTO**, to enable and disable the Autostart on the next Reset sequence. After an Autostart is enabled with a valid user vector, user application code will be started after Reset instead of the monitor or utility programs. To recover monitor operation after Autostart has been enabled, the **AUTO OFF** (**Spare option on Revision C boards**) option jumper can be installed or a low level applied to the XIRQ signal and Reset applied.

User application must perform all initialization including Stack setting, hardware startup, and external memory bus enable if needed, when the Autostart is applied. MON12 Ram Interrupt Vector table must also be applied in the same manner as under MON12 supervision or application interrupts will be trapped instead of serviced. See the CML12S.asm file for sample start-up initialization code.

Developing an application under MON12 or NOICE for Autostart should follow these steps:

1) Follow the MON12/ NOICE memory map and apply startup initialization and interrupt service routines in the 0x4000 - 0x7FFF memory area.

2) After development by applying ram memory program pages $20 - $2D, user should relocate the paged program code to internal flash pages $30 - $3D for programming into the flash memory. The user code in memory area 0x4000 - 0x7FFF will translate to the lower fixed flash page $3E for programming operations. User variables and stack, as well as interrupt vectors should stay in the internal ram area 0x1000 - 0x3E80 (monitor stack and variables not needed). Then set the Autostart vector for application launching.

3) If the Autostart application fails to start after programming, user should review all initialization and memory mapping first. Make sure the AUTO OFF (Spare) Option jumper is idle or open. If the application applies the XIRQ interrupt, the interrupt must be idle (high level) during any Reset sequence. Hardware may need to be applied if XIRQ signal level cannot be guaranteed high during Reset.

4) To perform a test Autostart and apply the external ram for program space the following precaution should be observed:

   Expanded Wide Mode bus operation must be enabled from internal Ram space before access to the external ram can be performed. Use CML12S.asm file for an example and locate the PEAR/MODE Register write in internal ram space 0x1000 - 0x3F80. Program pages $20 - $2D should be applied. Code must be loaded and tested without powering down the development board (use Reset Switch).

OPTIONS and JUMPERS

MEM_EN

The MEM_EN option jumper is installed by default and enables the external ram memory on the expanded HCS12 address and data bus. Removing the MEM_CS option jumper will allow single-chip I/O port type operation of HCS12 ports A, B, E, and K (no bus enabled) without external memory interference.

ECS

The ECS option installed enables the Emulation Chip Select signal from the HCS12 to drive the upper address lines from HCS12 Port K to the external Ram on the CML12S board. With the option open or idle, only the linear 64K byte address map is available on the external address / data bus. ECS installed is required to emulate flash program pages in the external ram memory.

MODC

The MODC option jumper provides Special Mode enable during Reset. This option must be open or idle when operating with the MON12 or NOICE monitors. If a BDM cable is applied to BDM port, the MODC option must be installed to enable Special Mode. Failure to install the MODC jumper during BDM application may cause communication problems with the host.

AUTO OFF / spare

The AUTO OFF (Spare on REV C. board) option jumper installed defeats the Autostart operation so the MON12 monitor will provide a command prompt. The jumper applies a ground potential to the XIRQ* interrupt line. User should only install the jumper to restore monitor operation, perform the MON12 NOAUTO command to disable the Autostart, and remove or idle the jumper. MON12 operation will then be provided on subsequent Reset conditions.

MODE

The MODE option jumper is not installed on the CML12S-DP256 board and is hard connected by circuit copper trace for Single-chip Mode operation of the CPU. Both the MODA and MODB signals are terminated by this option. Due to the restriction that the HCS12 internal flash memory is the only nonvolatile program memory provided on the board, single Chip Mode is default. All other Modes can be enabled under software control from this mode of operation.

The MODE option jumper may be installed by the user by cutting the hard trace and applying 2 header pins with a shunt jumper. With the shunt jumper removed, the Reset mode will then be Normal Expanded Wide. (Note: mask set 1K79X and earlier will not fetch the Reset vector from external memory in this mode).

OSC_SEL

The OSC_SEL option jumper is not installed on the CML12S-DP256 board. The default configuration is for the provided 4MHz reference crystal to provide the HCS12 oscillator. If the user requires an external clock to be applied, two header pins and shunt jumper can be applied to select the alternate clock source. User should refer to the HCS12 User Guide and CML12S board schematic for proper application of the external clock.

ROM_OFF

The ROM_OFF option jumper is not installed on the CML12S-DP256 board. The default configuration is that the internal flash memory of the HCS12 is enabled at Reset. The user must add external nonvolatile memory to the CML12S board to take advantage of this option. If the external memory is applied, the user may install the two header pins and shunt jumper to select internal or external memory use form Reset.

JP1 and JP2

JP1 and 2 option jumpers provide an easy method of connecting or isolating the HCS12 SCI0 and SCI1 serial channel RXD pins respectfully from the provided on-board RS232 transceiver. To apply the RXD pins on the SCI channels for other user applications requires that the transceiver driver be removed from the HCS12 pin. User may then apply signals to the respective pins at the MCU PORT connector without driver conflict. Please note that the on-board monitor(s) require HCS12 SCI channel 0 (JP1 installed) for user interface.

CUT-AWAY OPTIONS 1 - 6

CUT-AWAY options allow the user to disconnect dedicated HCS12 I/O port resources from development board peripherals. The CUT-AWAY options also allow for establishing the connection again by installing surface mount 1206 size 0 ohm resistors or mod wire with the use of a soldering iron. Normal operation of the development board generally does not require any manipulation of the CUT_AWAY options.

#1 Cut-Away: HCS12 Port S5/MOSI signal to the LCD_PORT shift register.

#2 Cut-Away: HCS12 Port S7/SS0 signal to the LCD_PORT shift register.

#3 Cut-Away: HCS12 Port S6/SCK signal to the LCD_PORT shift register.

#4 Cut-Away: HCS12 Oscillator Crystal ground, if another crystal is applied by the user this connection may require a capacitor to be installed. Refer to the HCS12 CGM module information.

#5 Cut-Away: HCS12 Port M0/CAN_RXD0 signal to the CAN port transceiver.

#6 Cut-Away: CAN Port Transceiver enable connection to ground. This connection enables the CAN Port transceiver output to the CAN bus at all times. If the user wants to apply output enable or slew rate control to the transceiver, this option should be cut and 1206 size resistor applied for slew rate or a HCS12 I/O port applied for output enable control. See the PCA82C250 data sheet for application information.

PORTS AND CONNECTORS

TB1 and J1 Power

The TB1 and J1 connectors provide power input to the board or if J1 is used for input, TB1 maybe used to source additional circuitry. The J1 power jack accepts a standard 2.0 ~ 2.1mm center barrel plug connector (positive voltage center) to provide the +VIN supply of +7 to +20 VDC @ 80ma minimum (+9VDC nominal). TB1 provides access to the +VIN, GND (power ground), HCS12 core VDD, and +5V power supplies. The CML12Sxxx power supply will provide 50ma of +5V for user application. +VIN input power should only be applied by J1 or TB1, not both or a supply conflict may occur and the CML12Sxxx board could be damaged. The VDD supply is for reference or external 2.5V input only and should not be loaded by external circuitry or damage to the HCS12 device may occur.

MCU_PORT

+5V	60	59	GND
PT7	58	57	PT6
PT5	56	55	PT4
PT3	54	53	PT2
PT1	52	51	PT0
** PK0	50	49	PK1 **
** PK2	48	47	PK3 **
** PK4	46	45	PK5 **
GND	44	43	+5V
PP1	42	41	PP0
PP3	40	39	PP2
PP5	38	37	PP4
PP7	36	35	PP6
** PM1	34	33	PM0 **
PM3	32	31	PM2
PM5	30	29	PM4
PM7	28	27	PM6
PJ1	26	25	PJ0
PJ7	24	23	PJ6
** PS7	22	21	PS6 **
** PS5	20	19	PS4 **
** PS3	18	17	PS2 **
** PS1	16	15	PS0 **
GND	14	13	+5V
GND	12	11	VREGEN
** PB7/D7	10	9	PB6/D6 **
** PB5/D5	8	7	PB4/D4 **
** PB3/D3	6	5	PB2/D2 **
** PB1/D1	4	3	PB0/D0 **
GND	2	1	+5V

The **MCU_PORT** provides access to the peripheral features and I/O lines of the HCS12.

** Note signals with alternate connections on the development board:

PB0 - 7 [D0 - 7] provide address / data on the expanded HCS12.

PK0 - 5 [XA14 - XA19] provide high order paged address lines on the expanded HCS12.

PM0 - 1 [CAN RXD0, TXD0] CAN channel 0 to CAN Port transceiver.

PS0 - 1 [COM Port RXD0, TXD0]

PS2 - 3 [JP3 Port RXD1, TXD2]

PS4 - 7 [SPI Port] provides LCD_PORT serial interface.

ANALOG PORT

PAD0/AN0	1	2	PAD8/AN8
PAD1/AN1	3	4	PAD9/AN9
PAD2/AN2	5	6	PAD10/AN10
PAD3/AN3	7	8	PAD11/AN11
PAD4/AN4	9	10	PAD12/AN12
PAD5/AN5	11	12	PAD13/AN13
PAD6/AN6	13	14	PAD14/AN14
PAD7/AN7	15	16	PAD15/AN15
VRH	17	18	VRL
VDDA	19	20	GND

The **ANALOG** port provides access to the Port AD0 and Port AD1 Analog-to-Digital input lines.

PAD0 – PAD15 HC12 Port AD0-15 is an input port or AN0 - AN15 A/D Converter inputs.

VRH / VRL HC12 A/D Converter Reference Pins. See HCS12 A/D User guide. To provide an external reference voltage, R3 and R4 need to be removed to apply external VRH or VRL respectfully. See schematic.

BUS_PORT

GND	1	2	D11/PA3
PA2/D10	3	4	D12/PA4
PA1/D9	5	6	D13/PA5
PA0/D8	7	8	D14/PA6
A0	9	10	D15/PA7
A1	11	12	A2
A10	13	14	A3
OE*	15	16	A4
A11	17	18	A5
A9	19	20	A6
A8	21	22	A7
A12	23	24	A13
WE*	25	26	A14
PE3/LSTRB*	27	28	A15
PE5/MODA	29	30	PE7/NOACC **
PE6/MODB	31	32	PE1/IRQ*
+5V	33	34	PE0/ XIRQ*
PE2/RW	35	36	RESERVED
PE4/ECLK	37	38	RESERVED
GND	39	40	RESET*

The **BUS_PORT** supports off-board memory devices while the HCS12 is in expanded mode.

PA0/D8 - PA7/D15 High Byte Data Bus in Wide Expanded Mode. Port A in Single Chip Mode.

A0 – A15 Latched Memory Addresses 0 to 15.

OE* Memory Output Enable signal, Active Low. Valid with ECLK and R/W high.

WE* Memory Write Enable signal, Active Low. Valid with ECLK high and R/W low.

RESET* HCS12 active low RESET signal.

KEYPAD / PORT H

The KEYPAD / PORT H connector provides interface for the HCS12 port H or applying a keypad such as the Axiom Mfg. HC-KP. When applied as a KEYPAD connector, the interface is for a passive 4 x 4 matrix (16 key) keypad device.

1	PH0
2	PH1
3	PH2
4	PH3
5	PH4
6	PH5
7	PH6
8	PH7

This interface is implemented as a software key scan. Pins PH0-3 are used as column drivers which are active high outputs. Pins PH4-7 are used for row input and will read high when their row is high.

See the file **`Key12Dx.ASM`** for an example program using this connector.

P_COM1 and P_COM2

1	1		6	
TXD0	2	6	7	
RXD0	3	7	8	
4	4	8	9	
GND	5	9	X	

The **COM-1** port has a Female DB9 connector that interfaces to the HCS12 internal SCI0 serial port via the U11 RS232 transceiver. It uses a simple 2 wire asynchronous serial interface and is translated to RS232 signaling levels.

1,4,6 connected and 7,8 connected

JP1 will isolate the SCI0 RXD pin from the transceiver.

The 1,4,6,7,8, and 9 pins provide RS232 flow control and status. These are connected on the on the bottom of the development board to provide NULL status to the host. User may isolate pins and provide flow control or status connection to the host by applying HCS12 I/O signals and **RS232 level conversion.**

P_COM2

P_COM2 is a 3 pin header that provides the HCS12 SCI1 serial port translated to RS232 signal levels. A solder cup DB9 style connector may be installed with wires and connector to apply this channel. JP2 option will isolate the SCI RXD pin from the transceiver.

P_COM2 pin connections:

Pin 1 = TXD

Pin 2 = RXD

Pin 3 = GND / common

CAN PORT

This port provides a CAN Bus interface associated with HCS12 CAN channel 0. The port has a CAN Transceiver (Philips PCA82C250) capable of up to 1M Baud data rate. The user may isolate the HCS12 CAN channel 0 from the transceiver by CUT-AWAY option 5.

CAN Port Connections

1	GND
2	CAN-H
3	CAN-L
4	+5V

The CAN Port connector provides an interface to the MSCAN12 channel 0 in the HCS12 microcontroller.

CAN BUS TRANSMIT ENABLE

The CAN port transceiver transmit driver is enabled for maximum drive and minimum slew rate by default. The drive and slew rate may be adjusted by cutting CUT-AWAY #6 and adding a 1206 size surface mount resistor, see the PCA82C250 data sheet for more information.

CAN Bus transceiver transmit enable control can be applied to the port by the RS tie pad. The user should select an available HCS12 I/O port to perform the transmit enable function and connect it from the MCU_PORT pin to RS pad as required. **The CUT_AWAY #6 must be open to apply transmit enable control.** The transmit enable signal to the CAN transceivers is active logic low.

CAN BUS TERMINATION

The CAN port provides RC11,12, and 13 1206 SMT size termination resistors on the bottom of the CML12Sxxx board that are not installed at the factory. The termination resistors provide optional bias and termination impedance for the CAN bus connected to the CAN Port. Type of wire media, data rate, length of wire, and number of CAN bus nodes can all effect the requirement or value of the termination for the CAN bus. User should refer to particular application for termination requirements.

RC11 CAN-H Bias Resistor: Provides bias to ground potential.

RC13 CAN-L Bias Resistor: Provides bias to +5V potential.

RC12 CAN Termination Resistor: Provides end point termination between CAN-H and CAN-L signal.

P1 - P4 HCS12 Header Ring

P1 - P4 provide a header ring for all I/O of the HCS12 device. These connectors are not installed. User should refer to the CML12S board schematic diagram for connector pin connections. All HCS12 I/O is available from the other I/O Ports on the board.

LCD_PORT

The LCD_PORT interface is connected to the HCS12 SPI-0 port and applies a serial shift register to convert the data to parallel interface for LCD input. This is required due to the fast timing characteristics of the HCS12 data bus and the slow timing of the standard LCD Modules. Example LCD Port assembly language driver software is provided on the support CD to demonstrate typical LCD module operation using this technique.

The interface supports all OPTREX™ DMC series and similar displays with up to 80 characters in 4 bit bus mode and provides the most common pinout for a dual row rear mounted display connector. The LCD module VEE or contrast potential is 0 Volts on this board. The LCD module type should be TN (Standard Twist) style and Reflective to support this VEE potential. The Axiom Mfg. HC-LCD is also compatible. The LCD Module is configured in a Write only mode, it is not possible to read current cursor position or the busy status back from the module.

LCD_PORT Connector

+5V	2	1	GND	SPI data bit definitions to LCD Port:
RS	4	3	VEE-GND	D0 - D3 = DB4 - 7, LCD data
EN	6	5	R/W-GND	D4 - D5 = Spare pins S1 and S2, not connected
DB1	8	7	DB0	D6 = RS, 0 = LCD Command, 1 = LCD Data
DB3	10	9	DB2	D7 = EN, 1 = LCD enable.
DB5	12	11	DB4	DB0 -DB3 are not applied and have 10K pull-down
DB7	14	13	DB6	resistance.

NOTES:

1) The LCD write requires 3 SPI transfers. Transfer 1 provides data 0 - 3 and RS (register select) value. Transfer 2 provides the same data with the EN (D7) bit set. Transfer 3 provides same data with the EN bit clear.

2) Resistor R25 can be removed to apply and external VEE potential.

3) CUT-AWAY 1 - 3 provide a means to isolate the LCD Port from the HCS12 SPI channel.

BDM PORT

The BDM port is a 6 pin header compatible with the Motorola Background Debug Mode (BDM) Pod. This allows the connection of a background debugger for software development, programming and debugging in real-time without using HC12 I/O resources.

BGND	1	2	GND	See the HC12 Technical Reference Manual for complete
	3	4	/RESET	documentation of the BDM.
	5	6	+5V	

A Background Debug Module is available from the manufacturer.

TEST POINTS

The following test points are provided on the development board:

EXTAL : HCS12 oscillator or external clock input pin.

XTAL : HCS12 oscillator output pin.

XFC : HCS12 PLL reference voltage and filter.

VDDPLL : HCS12 PLL voltage source test point.

TROUBLESHOOTING

The CML12SXXX board is fully tested and operational before shipping. If it fails to function properly, inspect the board for obvious physical damage first. Ensure that all IC devices in sockets are properly seated. Verify the communications setup as described under GETTING STARTED and see the **Tips and Suggestions** sections following for more information.

The most common problems are improperly configured communications parameters, and attempting to use the wrong COM port.

1. Verify that your communications port is working by substituting a known good serial device or by doing a loop back diagnostic.
2. Verify option jumpers JP1 is installed and MODC is open / idle.
3. Verify Autostart is not enabled. Apply a ground level to the XIRQ signal on the BUS_PORT and press the Reset switch. If the monitor prompts, erase the internal EEprom by performing a BULK command or disable the Autostart by following the procedure in the Autostart chapter.
4. Verify the power source. You should measure a minimum of 9 volts between the GND and +VIN connections on the TB1 power connector with the standard power supply provided.
5. If no voltage is found, verify the wall plug connections to 115VAC outlet and the power connector.
6. Verify the logic power source. You should measure +5 volts between the GND and +5V connections on the TB1 power connector. If the +VIN supply is good and this supply is not +5V, immediately disconnect power from the board. Contact support@axman.com by email for instructions and provide board name and problem.
7. Disconnect all external connections to the board except for COM1 to the PC and the wall plug.
8. Make sure that the RESET line is not being held low.
   Check for this by measuring the RESET Signal on the BUS_PORT.
9. Verify the presence of a 4MHz square wave at the EXTAL pin or 8MHz E clock signal if possible.
10. Contact support@axman.com by email for further assistance. Provide board name and describe problem.

Tips and Suggestions

Following are a number of tips, suggestions, and answers to common questions that will solve many problem users have with the CML12SXXX development system. You can download the latest software from the Support section of our web page at:

www.axman.com

Utilities

- If you're trying to program memory or start the utilities, make sure all jumpers are correct.
- Be certain that the data cable you're using is bi-directional and is connected securely to both the PC and the board. Also, make sure you are using the correct serial port.
- Make sure the correct power is supplied to the board. You should only use a 9 volt, 200mA minimum adapter or power supply. If you're using a power strip, make sure it is turned on.
- Make sure you load your code to an address space that actually exists. See the Memory Map if you're not sure. The MEM_EN and ECS options change the memory map.
- If debugging under Mon12, make sure you're not over-writing internal RAM used by it.
- If you're running in a multi-tasking environment (such as Windows™) close all programs in the background to be certain no serial conflict occurs.

Code Execution

- Under Mon12, breakpoints may not be acknowledged if you use the CALL command. You should use one of the GO command instead.
- Check the Autostart mask and reset vector located in EEprom at 0xFEC - 0xFEF. These 2 words contain the enable mask and address where user application execution will begin when the unit is powered on.
- When running your code stand-alone, you must initialize ALL peripherals used by the micro, including the Stack, Serial Port, and pseudo Interrupt vectors etc.
- You must either reset the COP watchdog timer in the main loop of your code or disable it when not running under Mon12 or BDM mode. The micro may enable this by default and if you don't handle it your code will reset every few 100ms.

TABLE 1: LCD Command and Character Codes

Command codes are used for LCD setup and control of character and cursor position. All command codes are written to LCD panel address $B5F0. The BUSY flag (bit 7) should be tested before any command updates to verify that any previous command is completed. A read of the command address $B5F0 will return the BUSY flag status and the current display character location address.

Command	Code	Delay
Clear Display, Cursor to Home	$01	1.65ms
Cursor to Home	$02	1.65ms
Entry Mode:		
Cursor Decrement, Shift off	$04	40us
Cursor Decrement, Shift on	$05	40us
Cursor Increment, Shift off	$06	40us
Cursor Increment, Shift on	$07	40us
Display Control:		
Display, Cursor, and Cursor Blink off	$08	40us
Display on, Cursor and Cursor Blink off	$0C	40us
Display and Cursor on, Cursor Blink off	$0E	40us
Display, Cursor, and Cursor Blink on	$0F	40us
Cursor / Display Shift: (nondestructive move)		
Cursor shift left	$10	40us
Cursor shift right	$14	40us
Display shift left	$18	40us
Display shift right	$1C	40us
Display Function (default 2x40 size)	$3C	40us
Character Generator Ram Address set	$40-$7F	40us
Display Ram Address and set cursor location	$80- $FF	40us

LCD Character Codes

$20	Space	$2D	-	$3A	:	$47	G	$54	T	$61	A	$8E	n	$7B	{
$21	!	$2E	.	$3B	;	$48	H	$55	U	$62	B	$8F	o	$7C	|
$22	"	$2F	/	$3C	<	$49	I	$56	V	$63	C	$70	p	$7D	}
$23	#	$30	0	$3D	=	$4A	J	$57	W	$64	D	$71	q	$7E	>
$24	$	$31	1	$3E	>	$4B	K	$58	X	$65	E	$72	r	$7F	<
$25	%	$32	2	$3F	?	$4C	L	$59	Y	$66	F	$73	s		
$26	&	$33	3	$40	Time	$4D	M	$5A	Z	$67	G	$74	t		
$27	'	$34	4	$41	A	$4E	N	$5B	[	$68	H	$75	u		
$28	(	$35	5	$42	B	$4F	O	$5C	Yen	$69	I	$76	v		
$29	)	$36	6	$43	C	$50	P	$5D	]	$8A	J	$77	w		
$2A	*	$37	7	$44	D	$51	Q	$5E	^	$8B	K	$78	x		
$2B	+	$38	8	$45	E	$52	R	$5F	_	$6C	L	$79	y		
$2C	,	$39	9	$48	F	$53	S	$80	`	$8D	M	$7A	z		

TABLE 2: MON12 Service Routine Jump Table

```
ADDRESS
ff10    JMP     MAIN            ; warm start
ff13    JMP     BPCLR           ; clear breakpoint table
ff16    JMP     RPRINT          ; display user registers
ff19    JMP     HEXBIN          ; convert ascii hex char to binary
ff1c    JMP     BUFFARG         ; build hex argument from buffer
ff1f    JMP     TERMARG         ; read hex argument from terminal
ff22    JMP     CHGBYT          ; modify memory byte at address in x
ff25    JMP     CHGWORD         ; modify memory word at address in x
ff28    JMP     READBUFF        ; read character from buffer
ff2b    JMP     INCBUFF         ; increment buffer pointer
ff2e    JMP     DECBUFF         ; decrement buffer pointer
ff31    JMP     WSKIP           ; find non-whitespace char in buffer
ff34    JMP     CHKABRT         ; check for abort from terminal
ff37    JMP     UPCASE          ; convert to upper case
ff3a    JMP     WCHEK           ; check for white space
ff3d    JMP     DCHEK           ; check for delimeter
ff40    JMP     ONSCI0          ; initialize i/o device
ff43    JMP     INPUT           ; low level input routine
ff46    JMP     OUTPUT          ; low level output routine
ff49    JMP     OUTLHLF         ; display top 4 bits as hex digit
ff4c    JMP     OUTRHLF         ; display bottom 4 bits as hex digit
ff4f    JMP     OUTA            ; output ascii character in A
ff52    JMP     OUT1BYT         ; display the hex value of byte at X
ff55    JMP     OUT1BSP         ; outlbyt followed by space
ff58    JMP     OUT2BSP         ; display 2 hex bytes (word) at x and a space
ff5b    JMP     OUTCRLF         ; carriage return, line feed to terminal
ff5e    JMP     OUTSTRG         ; display string at X (term with $04)
ff61    JMP     OUTSTRG0        ; outstrg with no initial carr ret
ff64    JMP     INCHAR          ; wait for and input a char from term
ff67    JMP     VECINIT         ; initialize RAM vector table
```

TABLE 3: MON12 Interrupt Table

MON12 Ram Interrupt Vector	HCS12 Interrupt Vector Address	MON12 TRAP code	Vector Source
3F8C	FF8C	02	PWME
3F8E	FF8E	04	PTPI
3F90	FF90	06	C4TX
3F92	FF92	08	C4RX
3F94	FF94	0A	C4ERR
3F96	FF96	0C	C4WU
3F98	FF98	0E	C3TX
3F9A	FF9A	10	C3RX
3F9C	FF9C	12	C3ERR
3F9E	FF9E	14	C3WU
3FA0	FFA0	16	C2TX
3FA2	FFA2	18	C2RX
3FA4	FFA4	1A	C2ERR
3FA6	FFA6	1C	C2WU
3FA8	FFA8	1E	C1TX
3FAA	FFAA	20	C1RX
3FAC	FFAC	22	C1ERR
3FAE	FFAE	24	C1WU
3FB0	FFB0	26	C0TX
3FB2	FFB2	28	C0RX
3FB4	FFB4	2A	C0ERR
3FB6	FFB6	2C	C0WU
3FB8	FFB8	2E	FEPRG
3FBA	FFBA	30	EEPRG
3FBC	FFBC	32	SPI2
3FBE	FFBE	34	SPI1
3FC0	FFC0	36	I2C
3FC2	FFC2	38	BDLC
3FC4	FFC4	3A	CRGC
3FC6	FFC6	3C	CRGL
3FC8	FFC8	3E	PACBO
3FCA	FFCA	40	MCNT
3FCC	FFCC	42	PTHI
3FCE	FFCE	44	PTJI
3FD0	FFD0	46	ADC1
3FD2	FFD2	48	ADC0
3FD4	FFD4	4A	SCI1
3FD6	FFD6	4C	SCI0
3FD8	FFD8	4E	SPI0
3FDA	FFDA	50	PACAI
3FDC	FFDC	52	PACAO
3FDE	FFDE	54	TOF
3FE0	FFE0	56	TC7
3FE2	FFE2	58	TC6
3FE4	FFE4	5A	TC5
3FE6	FFE6	5C	TC4
3FE8	FFE8	5E	TC3
3FEA	FFEA	60	TC2
3FEC	FFEC	62	TC1
3FEE	FFEE	64	TC0
3FF0	FFF0	66	RTI
3FF2	FFF2	68	IRQ
3FF4	FFF4	6A	XIRQ
3FF6	FFF6	6C	SWI
3FF8	FFF8	6E	TRAP
3FFA	FFFA	70	COP
3FFC	FFFC	72	CLM
	FFFE		RESET

CML-9S12DP512

Description: Low cost development system for the MC9S12DP512

Product Image:

Get original file (61KB)

Product Summary: The CML-912SDP512 is a low cost development system for the Motorola MC9S12DP512 Microcontroller. The board provides operation in Single-Chip Mode with the Expanded Wide Bus available for expansion and development memory access. The system is supplied with Monitor / Debugger installed with programming support utilities. Features provide for easy selection of operation mode, flash programming, BDM operation, keypad and LCD module connections, and prototyping. All required power supplies and support software is included to complete and program an application.

Features: • MC9S12DP512 Features:

Upward code compatible w/ 68HC11
4K Bytes EEPROM
512K Byte Flash EEPROM
14K Byte SRAM
2 Enhanced SCI Ports
3 SPI Port (Synchronous Serial)
5 CAN 2.0 A or B Interface
Two 8 Channel 10 Bit Analog Converters
Background Debug Port
Enhanced 16 bit Timer w/ 8 channels
of capture or compare
16 Bit Pulse Accumulator
8 PWM Channels
Two 8 bit Key Wake-up ports

PLL Clock Oscillator Support
RTC and COP features
Up to 91 I/O
• 4Mhz reference oscillator for up to 24MHz operation.
• External Memory: 256K Bytes (128K x 16) SRAM
• COM1 Port HC12 SCI0 w/ RS232 and DB9S connector
• COM2 Port HC12 SCI1 w/ RS232 and 3 pin header
• INDICATORS Power and RESET.
• BUS-PORT 40 Pin Socket Header
• MCU I/O PORT - 60 pin Socket Header
• Analog Port 20 pin Socket Header
• CAN PORT CAN 0 I/O with 1M Baud Transceiver
• LCD Module and Keypad Ports
• Solderless Prototype Area and Connections
• Easy Power Connection and Tap points
• Back Ground Debug (BDM) Port 6 Pin standard

Specifications: • 7 to 25VDC input to 5V Power Supply
• Operating Power: 60ma @ 5V
Package Contents: The Axiom development system provides for low cost software debugging with the use of a Debug Monitor installed in the internal or external memory. Operation allows the user to locate code in the On-Board RAM, set Break Points, Trace, and display or modify registers or memory. After code is operational the user may relocate the code and program the MC9S12DP512 internal Flash for dedicated operation of new software. No additional hardware or software is required. Board is compatible with standard HC12 BDM pods and software compilers that provide an integrated debug interface.

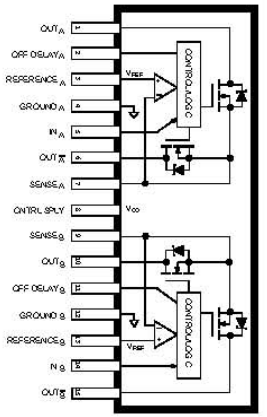
Appendix H: Motor Driver Datasheet

SLA7024M, SLA7026M, AND SMA7029M

Data Sheet 29201

HIGH-CURRENT PWM, UNIPOLAR STEPPER MOTOR CONTROLLER/DRIVERS

SMA7029M



Dwg. PK-007

ABSOLUTE MAXIMUM RATINGS
at T_A = +25°C

Load Supply Voltage, V_{BB} . . . . . 46 V
FET Output Voltage, V_{DS} . . . . . 100 V
Control Supply Voltage, V_{CC} . . . . . 46 V
Peak Output Current, I_{OUTM} ($t_w \leq 100$ µs)
  SLA7024M . . . . . 3.0 A
  SLA7026M . . . . . 5.0 A
  SMA7029M . . . . . 3.0 A
Continuous Output Current, I_{OUT}
  SLA7024M . . . . . 1.5 A
  SLA7026M . . . . . 3.0 A
  SMA7029M . . . . . 1.5 A
Input Voltage Range, V_{IN} . . . . -0.3 V to 7.0 V
Reference Voltage, V_{REF} . . . . . 2.0 V
Package Power Dissipation, P_D . See Graph
Junction Temperature, T_J . . . . . +150°C
Operating Temperature Range,
  T_A . . . . . -20°C to +85°C
Storage Temperature Range,
  T_{stg} . . . . . -40°C to +150°C

The SLA7024M, SLA7026M, and SMA7029M are designed for high-efficiency and high-performance operation of 2-phase, unipolar stepper motors. An automated, innovative packaging technology combined with power FETs and monolithic logic/control circuitry advances power multi-chip modules (PMCMs™) toward the complete integration of motion control. Highly automated manufacturing techniques provide low-cost and exceptionally reliable PMCMs suitable for controlling and directly driving a broad range of 2-phase, unipolar stepper motors. The three stepper motor multi-chip modules differ primarily in output current ratings (1.5 A or 3.0 A) and package style.

All three PMCMs are rated for an absolute maximum limit of 46 V and utilize advanced NMOS FETs for the high-current, high-voltage driver outputs. The avalanche-rated (≥100 V) FETs provide excellent ON resistance, improved body diodes, and very-fast switching. The multi-chip ratings and performance afford significant benefits and advantages for stepper drives when compared to the higher dissipation and slower switching speeds associated with bipolar transistors. Normally, heat sinks are not required for the SLA7024M or SMA7029M. The SLA7026M, in demanding, higher-current systems designs, necessitates suitable heat transfer methods for reliable operation.

Complete applications information is given on the following pages. PWM current is regulated by appropriately choosing current-sensing resistors, a voltage reference, a voltage divider, and RC timing networks. The RC components limit the OFF interval and control current decay. Inputs are compatible with 5 V logic and microprocessors.

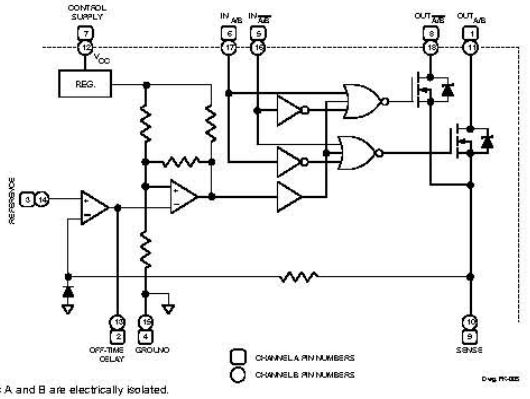
BENEFITS AND FEATURES

- Cost-Effective, Multi-Chip Solution
- 'Turn-Key' Motion-Control Module
- Motor Operation to 3 A and 46 V
- 3rd Generation High-Voltage FETs
- 100 V, Avalanche-Rated NMOS
- Low $r_{DS(on)}$ NMOS Outputs
- Advanced, Improved Body Diodes
- Single-Supply Motor/Module Operation
- Half- or Full-Step Unipolar Drive
- High-Efficiency, High-Speed PWM
- Dual PWM Current Control (2-Phase)
- Programmable PWM Current Control
- Low Component Count PWM Drive
- Low Internal Power Dissipation
- Heat Sinking (Normally) Unnecessary
- Electrically Isolated Power Tab
- Logic IC- and µP-Compatible Inputs
- Machine-Insertable Package

Always order by complete part number:

Part Number	Package	Output Current
SLA7024M	18-Lead Power-Tab SIP	1.5 A
SLA7026M	18-Lead Power-Tab SIP	3.0 A
SMA7029M	15-Lead SIP	1.5 A

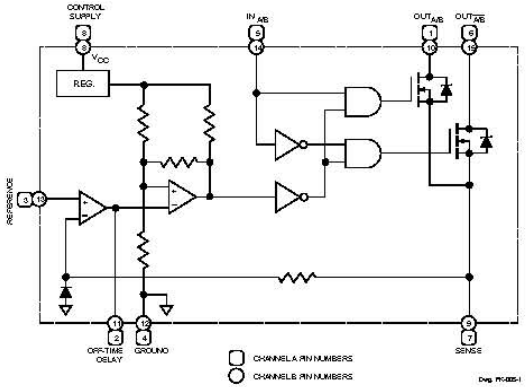
SLA7024M and SLA7026M FUNCTIONAL BLOCK DIAGRAM



Dwg. PK-005

Note that channels A and B are electrically isolated.

SMA7029M FUNCTIONAL BLOCK DIAGRAM

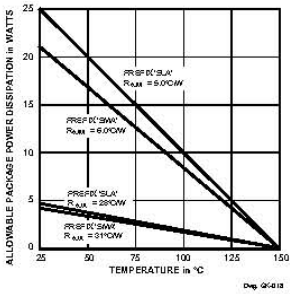


Dwg. PK-005-1

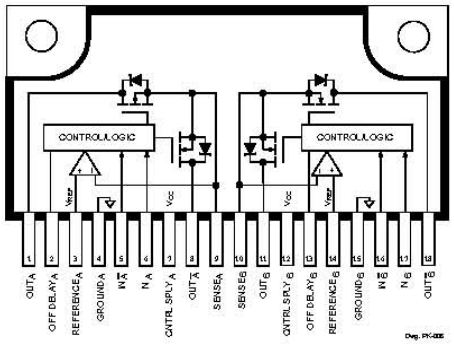
Note that except for the control supply, channels A and B are electrically isolated.

115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

ALLOWABLE PACKAGE POWER DISSIPATION



SLA7024M and SLA7026M



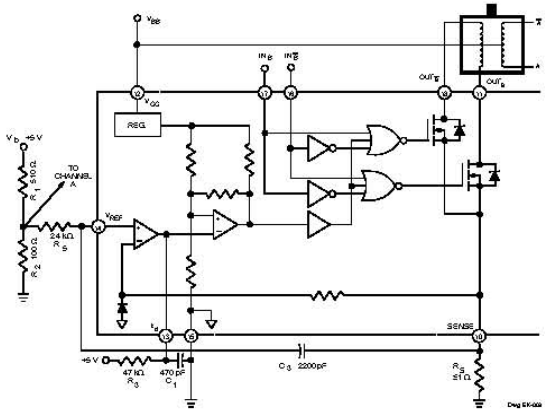
ELECTRICAL CHARACTERISTICS at T_A = +25°C

Characteristic	Symbol	Test Conditions	Limits			Units
			Min	Typ	Max	
FET Leakage Current	I_{DSS}	V_{DS} = 100 V, V_{CC} = 44 V			4.0	mA
FET ON Voltage	$V_{DS(ON)}$	(SLA7024M & SMA7029M) V_{CC} = 14 V, I_{OUT} = 1 A			600	mV
		(SLA7026M) V_{CC} = 14 V, I_{OUT} = 3 A			850	mV
FET ON Resistance	$r_{DS(on)}$	(SLA7024M & SMA7029M) V_{CC} = 14 V, I_{OUT} = 1 A			600	mΩ
		(SLA7026M) V_{CC} = 14 V, I_{OUT} = 3 A			285	mΩ
Body Diode Forward Voltage	V_{SD}	(SLA7024M & SMA7029M) I_{OUT} = 1 A		0.9	1.5	V
		(SLA7026M) I_{OUT} = 3 A		0.9	1.6	V
Control Supply Voltage	V_{CC}	Operating	10	24	44	V
Control Supply Current	I_{CC}	V_{CC} = 44 V		10	15	mA
Input Current	$I_{IN(H)}$	V_{CC} = 44 V, V_{IN} = 2.4 V			40	µA
	$I_{IN(L)}$	V_{IN} = 0.4 V			-800	µA
Input Voltage	$V_{IN(H)}$		2.0			V
	$V_{IN(L)}$				0.8	V

NOTE: Negative current is defined as coming out of (sourcing) the specified device pin.

SLA7024M, SLA7026M, AND SMA7029M
HIGH-CURRENT PWM,
UNIPOLAR STEPPER MOTOR
CONTROLLER/DRIVERS

TYPICAL STEPPER MOTOR APPLICATIONS
(Half of Each Device Shown)
SLA7024M and SLA7026M



TRUTH TABLES
(Device Types as Designated)

WAVE DRIVE (FULL STEP) for SLA7024M and SLA7026M

Sequence	0	1	2	3	0
Input A	H	L	L	L	H
Input $\overline{A}$	L	L	H	L	L
Input B	L	H	L	L	L
Input $\overline{B}$	L	L	L	H	L
Output ON	A	B	$\overline{A}$	$\overline{B}$	A

2-PHASE (FULL STEP) OPERATION for SLA7024M and SLA7026M

Sequence	0	1	2	3	0
Input A	H	L	L	H	H
Input $\overline{A}$	L	H	H	L	L
Input B	H	H	L	L	H
Input $\overline{B}$	L	L	H	H	L
Outputs ON	AB	$\overline{A}$ B	$\overline{AB}$	A $\overline{B}$	AB

HALF-STEP OPERATION (2-1-2 SEQUENCE) for SLA7024M, SLA7026M, and SMA7029M

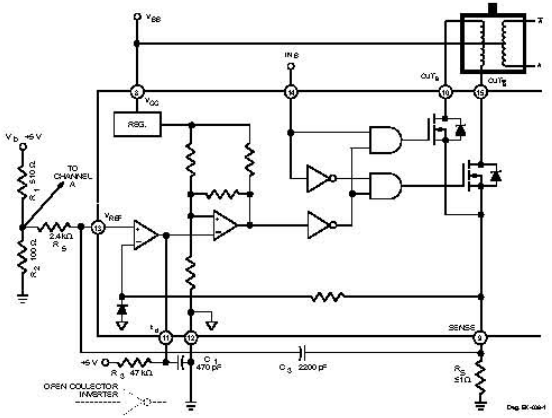
Sequence	0	1	2	3	4	5	6	7	0
Input A	H	H	L	L	L	L	L	H	H
Input $\overline{A}$ or t_{dA}*	L	L	L	H	H	H	L	L	L
Input B	L	H	H	H	L	L	L	L	L
Input $\overline{B}$ or t_{dB}*	L	L	L	L	L	H	H	H	L
Output(s) ON	A	AB	B	$\overline{A}$ B	$\overline{A}$	$\overline{AB}$	$\overline{B}$	A $\overline{B}$	A

*Logic signals to external open-collector inverter connected to t_{dA} and t_{dB}.

115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

TYPICAL STEPPER MOTOR APPLICATIONS

(Half of Device Shown)
SMA7029M



TRUTH TABLES

(SMA7029M Only)

WAVE DRIVE (FULL STEP) for SMA7029M

Sequence	0	1	2	3	0
Input A	H	L	L	L	H
Input tdA*	L	L	H	L	L
Input B	L	H	L	L	L
Input tdB*	L	L	L	H	L
Output ON	A	B	$\overline{A}$	$\overline{B}$	A

*Logic signals to external open-collector inverter connected to t_{dA} and t_{dB}.

2- PHASE (FULL STEP) OPERATION for SMA7029M

Sequence	0	1	2	3	0
Input A	H	H	L	L	H
Input B	L	H	H	L	L
Outputs ON	A B	AB	A B	$\overline{AB}$	A B

SLA7024M, SLA7026M, AND SMA7029M
HIGH-CURRENT PWM,
UNIPOLAR STEPPER MOTOR
CONTROLLER/DRIVERS

APPLICATIONS INFORMATION

REGULATING THE PWM OUTPUT CURRENT

The output current (and motor coil current) waveform is illustrated in Figure 1. Setting the PWM current trip point requires various external components:

V_b = Reference supply (typically 5 V)

R_1, R_2 = Voltage-divider resistors in the reference supply circuit

R_S = Current sensing resistor(s)

NOTE: The maximum allowable V_{REF} input voltage is 2.0 V. The voltage-divider must be selected accordingly.

Normal PWM (Full-Current/Running) Mode

I_{OUT} is set to meet the specified running current for the motor (Figure 2) and is determined by:

$$I_{OUT} \approx \frac{V_{REF}}{R_S} \quad (1)$$

or, if V_{REF} is not known

$$I_{OUT} \approx \frac{R_2}{R_1 + R_2} \cdot \frac{V_b}{R_S} \quad (2)$$

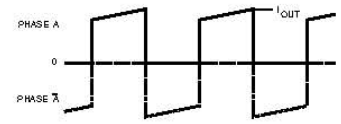


FIGURE 1. PHASE A COIL CURRENT WAVEFORM

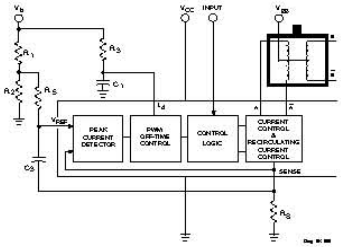


FIGURE 2. PWM CONTROL (RUN MODE)

115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

For given values of R_1, R_2, and V_b (V_{REF} = 0.82 V), Figure 3 illustrates output current as a function of current-sensing resistance (R_S).

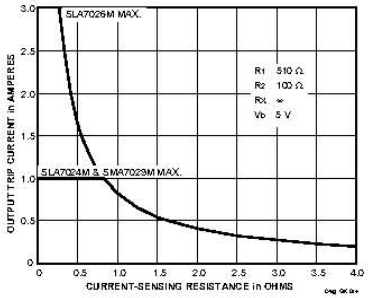


FIGURE 3. CURRENT-SENSING RESISTANCE

Reduced/Holding Current Mode

Additional circuitry (Figure 4) enables reducing motor current. The external transistor changes the voltage-divider ratio, V_{REF}, and reduces the output current. I_{HOLD} is determined by resistors R_2 and R_X in parallel:

$$I_{HOLD} = \frac{R_2 R_X}{R_1 R_2 + R_1 R_X + R_2 R_X} \cdot \frac{V_b}{R_S} \qquad (3)$$

or

$$I_{HOLD} = \frac{R_2 \ddot{O}}{R_1 + R_2 \ddot{O}} \cdot \frac{V_b}{R_S} \qquad (4)$$

where $R_2 \ddot{O}$ = the equivalent value of R_2 and R_X in parallel.

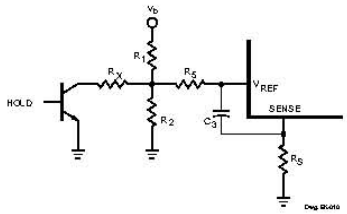


FIGURE 4. HOLD CURRENT MODE

SLA7024M, SLA7026M, AND SMA7029M
HIGH-CURRENT PWM,
UNIPOLAR STEPPER MOTOR
CONTROLLER/DRIVERS

For given values of R_1, R_2, and V_b (V_{REF} = 0.82 V), Figures 5A and 5B illustrate output holding current as a function of R_X for two values of current-sensing resistance (R_S).

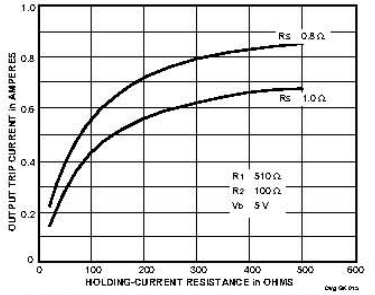


FIGURE 5A. HOLD-CURRENT RESISTANCE (SLA7024M and SMA7029M)

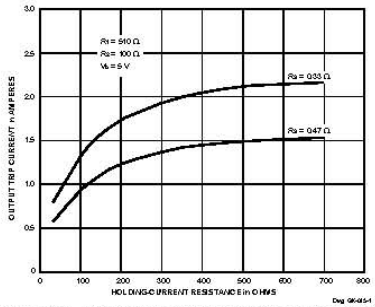


FIGURE 5B. HOLD-CURRENT RESISTANCE (SLA7026M)

NOTE: Holding current determines holding torque, which is normally greater than running torque. Consult motor manufacturer for recommended safe holding current and motor winding temperature limits in "standstill" or "detent" mode.

The MOSFET outputs create ringing noise with PWM, but the RC filter precludes malfunctions. The comparator operation is affected by R_S and C_3 and, thus, current overshoot is influenced by component values. Empirical adjustment to "fine-tune" the current limit is likely.

115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

DETERMINING THE MOTOR PWM FREQUENCY

The modules function asynchronously, with PWM OFF time fixed by R_g and C_t at input t_g. The OFF time can be calculated as:

$$t_{OFF} = -R_g \cdot C_t \cdot \log_n \left(1 - \frac{2}{V_b}\right) \quad (5)$$

Recommended circuit constants and t_{OFF} are:

V_b = 5 V
R_g = 47 kΩ
C_t = 470 pF
t_{OFF} = 12 µs

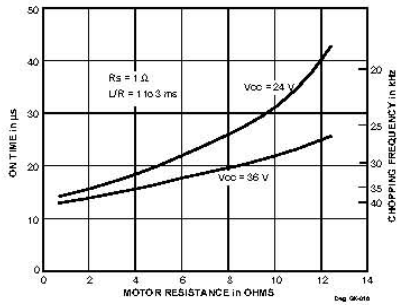


FIGURE 7.
PWM FREQUENCY vs MOTOR RESISTANCE

POWER DISSIPATION CALCULATIONS

Excepting high-current applications utilizing the SLA7026M above approximately 2.0 A at +65°C (with 2-phase operation), the need for heat sinks is rare. The basic constituents of conduction losses (internal power dissipation) include:

(a) FET output power dissipation ($I_{OUT}^2 \cdot r_{DS(on)}$ or $I_{OUT} \cdot V_{DS(ON)}$),
(b) FET body diode power dissipation ($V_{SD} \cdot I_{OUT}$), and
(c) control circuit power dissipation ($V_{CC} \cdot I_{CC}$).

Device conduction losses are calculated based on the operating mode (wave drive, half-step, or 2-phase). Assuming a 50% output duty cycle:

Wave Drive = 0.5 ($I_{OUT}^2 \cdot r_{DS(on)}$) + 0.5 ($V_{SD} \cdot I_{OUT}$) + ($V_{CC} \cdot$ 15 mA)
Half-Step = 0.75 ($I_{OUT}^2 \cdot r_{DS(on)}$) + 0.75 ($V_{SD} \cdot I_{OUT}$) + ($V_{CC} \cdot$ 15 mA)
2-Phase = ($I_{OUT}^2 \cdot r_{DS(on)}$) + ($V_{SD} \cdot I_{OUT}$) + ($V_{CC} \cdot$ 15 mA)

SLA7024M, SLA7026M, AND SMA7029M
HIGH-CURRENT PWM, UNIPOLAR STEPPER MOTOR CONTROLLER/DRIVERS

PACKAGE RATINGS/DERATING FACTORS

Thermal ratings/deratings for the multi-chip module packages vary slightly. Normally, the SLA7024M and SMA7029M do not need heat sinking when operated within maximum specified output current (≤1.0 A with 2-phase drive) unless the design ambient temperature also exceeds +60°C. Thermal calculations must also consider the temperature effects on the output FET ON resistance. The applicable thermal ratings for the PMCM packages are:

SLA7024M and SLA7026M 18-Lead Power-Tab SIP

$R_{\theta JA}$ = 28°C/W (no heat sink) or 4.5 W at +25°C and a derating factor of -36 mW/°C for operation above +25°C. $R_{\theta JC}$ = 5°C/W.

SMA7029M 15-Lead SIP

$R_{\theta JA}$ = 31°C/W (no heat sink) or 4.0 W at +25°C and a derating factor of -32 mW/°C for operation above +25°C. $R_{\theta JC}$ = 6°C/W.

TEMPERATURE EFFECTS ON FET $r_{DS(on)}$

Analyzing safe, reliable operation includes a concern for the relationship of NMOS ON resistance to junction temperature. Device package power calculations must include the increase in ON resistance (producing higher output ON voltages) caused by higher operating junction temperatures. Figure 8 provides a normalized ON resistance curve, and all thermal calculations should consider increases from the given +25°C limits, which may be caused by internal heating during normal operation.

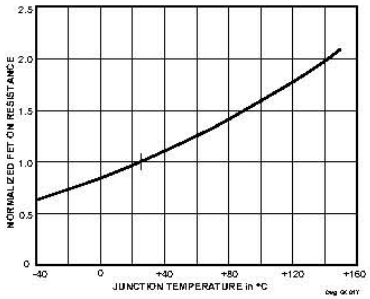
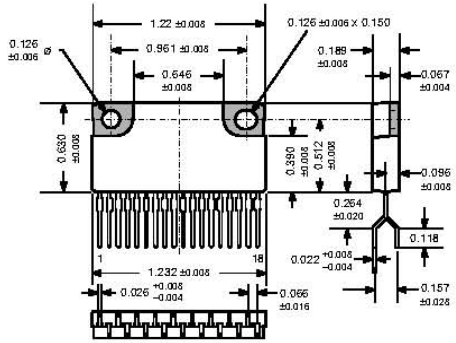


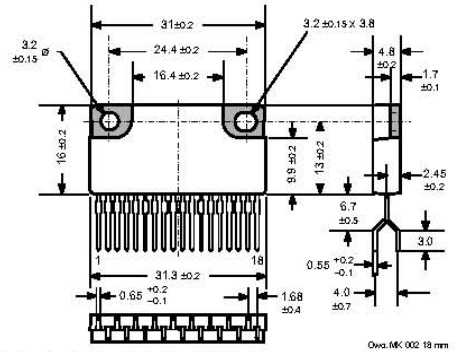
FIGURE 8. NORMALIZED ON RESISTANCE vs TEMPERATURE

115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

SLA7024M and SLA7026M
Dimensions in Inches
(for reference only)



Dimensions in Millimeters
(controlling dimensions)

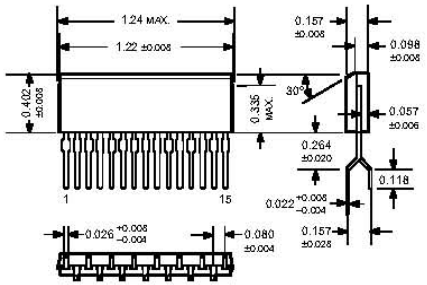


Dwg. MK-002-18-mm

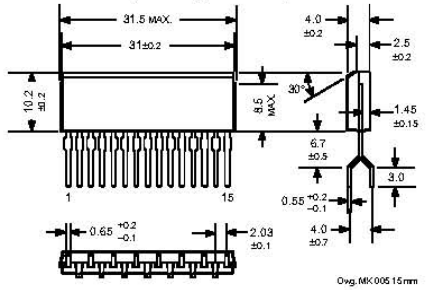
NOTES: 1. Exact body and lead configuration at vendor's option within limits shown.
2. Recommended mounting hardware torque: 4.34 – 5.79 lbf•ft (6 – 8 kgf•cm or 0.588 – 0.784 Nm).
3. The hatched area is exposed (electrically isolated) heat spreader.
4. Recommend use of metal-oxide-filled, alkyl-degenerated oil base, silicone grease (Dow Corning 340 or equivalent).

SLA7024M, SLA7026M, AND SMA7029M
HIGH-CURRENT PWM,
UNIPOLAR STEPPER MOTOR
CONTROLLER/DRIVERS

SMA7029M
Dimensions in Inches
(for reference only)



Dimensions in Millimeters
(controlling dimensions)



Dwg. MK-005-15-mm

NOTE: Exact body and lead configuration at vendor's option within limits shown.

115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

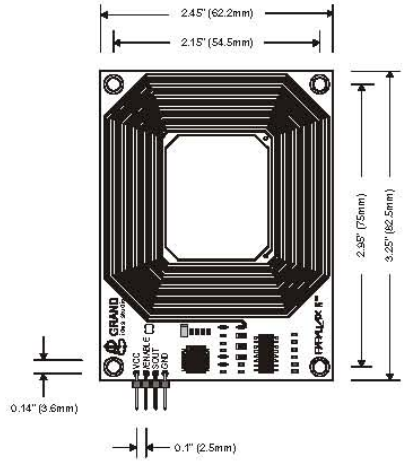
Appendix I: RFID Datasheet

PARALLAX
599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

General: info@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com
Educational: www.stampsinclass.com

RFID Reader Module (#28140)

RFID 54 mm x 85 mm Rectangle Tag (#28141)
RFID 50 mm Round Tag (#28142)



Introduction

Designed in cooperation with Grand Idea Studio (http://www.grandideastudio.com/), the Parallax Radio Frequency Identification (RFID) Reader Module is the first low-cost solution to read passive RFID transponder tags up to 1 ¾" - 3" inches away depending on the tag (see list below). The RFID Reader Module can be used in a wide variety of hobbyist and commercial applications, including access control, automatic identification, robotics navigation, inventory tracking, payment systems, and car immobilization.

- Fully-integrated, low-cost method of reading passive RFID transponder tags
- 1-wire, 2400 baud Serial TTL interface to PC, BASIC Stamp® and other processors
- Requires single +5VDC supply
- Bi-color LED for visual indication of activity
- 0.100" pin spacing for easy prototyping and integration

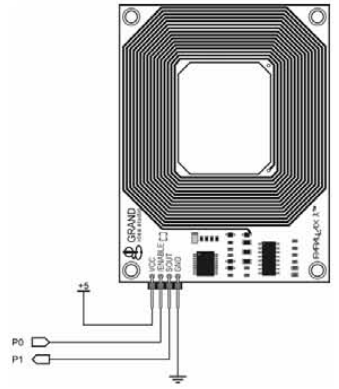
The Parallax RFID Reader Module works exclusively with the EM Microelectronics-Marin SA EM4100-family of passive read-only transponder tags. A variety of different tag types and styles exist with the most popular made available from Parallax. Each transponder tag contains a unique identifier (one of 2^{40}, or 1,099,511,627,776, possible combinations) that is read by the RFID Reader Module and transmitted to the host via a simple serial interface.

Electronic Connections

The Parallax RFID Reader Module can be integrated into any design using only four connections (VCC, /ENABLE, SOUT, GND). Use the following circuit for connecting the Parallax RFID Reader Module to the BASIC Stamp microcontroller:

Pin	Pin Name	Type	Function
1	VCC	P	System power, +5V DC input.
2	/ENABLE	I	Module enable pin. Active LOW digital input. Bring this pin LOW to enable the RFID reader and activate the antenna.
3	SOUT	O	Serial Out. TTL-level interface, 2400bps, 8 data bits, no parity, 1 stop bit.
4	GND	G	System ground. Connect to power supply's ground (GND) terminal.

Note: Type: I = Input, O = Output, P = Power, G = Ground



Communication Protocol

Implementation and usage of the RFID Reader Module is straightforward. BASIC Stamp 1, 2, and SX28AC/DP code examples (SX/B) are included at the end of this documentation.

The RFID Reader Module is controlled with a single TTL-level active-low /ENABLE pin. When the /ENABLE pin is pulled LOW, the module will enter its active state and enable the antenna to interrogate for tags. The current consumption of the module will increase dramatically when the module is active.

A visual indication of the state of the RFID Reader Module is given with the on-board LED. When the module is successfully powered-up and is in an idle state, the LED will be GREEN. When the module is in an active state and the antenna is transmitting, the LED will be RED.

The face of the RFID tag should be held parallel to the front or back face of the antenna (where the majority of RF energy is focused). If the tag is held sideways (perpendicular to the antenna) you'll either get no reading or a poor reading. Only one transponder tag should be held up to the antenna at any time. The use of multiple tags at one time will cause tag collisions and confuse the reader. The two tags available in the Parallax store have a read distance of approximately 3 inches. Actual distance may vary slightly depending on the size of the transponder tag and environmental conditions of the application.

When a valid RFID transponder tag is placed within range of the activated reader, the unique ID will be transmitted as a 12-byte ASCII string via the TTL-level SOUT (Serial Output) pin in the following format:

MSB LSB

Start Byte (0x0A)	Unique ID Digit 1	Unique ID Digit 2	Unique ID Digit 3	Unique ID Digit 4	Unique ID Digit 5	Unique ID Digit 6	Unique ID Digit 7	Unique ID Digit 8	Unique ID Digit 9	Unique ID Digit 10	Stop Byte (0x0D)

The start byte and stop byte are used to easily identify that a correct string has been received from the reader (they correspond to a line feed and carriage return characters, respectively). The middle ten bytes are the actual tag's unique ID.

All communication is 8 data bits, no parity, 1 stop bit, non-inverted, least significant bit first (8N1). The baud rate is configured for 2400bps, a standard communications speed supported by most any microprocessor or PC, and cannot be changed. The Parallax RFID Reader Module initiates all communication. The Parallax RFID Reader Module can connect directly to any TTL-compatible UART or to an RS232-compatible interface by using an external level shifter.

Absolute Maximum Ratings and Electrical Characteristics

Condition	Value
Operating Temperature	-40°C to +85°C
Storage Temperature	-55°C to +125°C
Supply Voltage (V_{CC})	+4.5V to +5.5V
Ground Voltage (V_{SS})	0V
Voltage on any pin with respect to V_{SS}	-0.3V to +7.0V

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

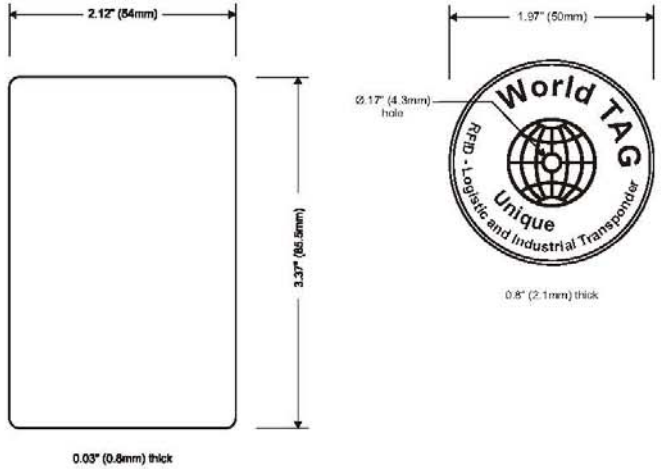
At V_{CC} = +5.0V and T_A = 25°C unless otherwise noted

Parameter	Symbol	Test Conditions	Specification Min.	Typ.	Max.	Unit
Supply Voltage	V_{CC}	—	4.5	5.0	5.5	V
Supply Current, Idle	I_{IDLE}	—	—	10	—	mA
Supply Current, Active	I_{CC}	—	—	90	—	mA
Input LOW voltage	V_{IL}	+4.5V <= V_{CC} <= +5.5V	—	—	0.8	V
Input HIGH voltage	V_{IH}	+4.5V <= V_{CC} <= +5.5V	2.0	—	—	V
Output LOW voltage	V_{OL}	V_{CC} = +4.5V	—	—	0.8	V
Output HIGH voltage	V_{OH}	V_{CC} = +4.5V	V_{CC} - 0.7	—	—	V

RFID Tags Available From Parallax

Parallax provides two passive RFID tags from our on-line store. We're stocking the tags because many suppliers have high minimums, yet many of our customers may only want a few tags for their basic experimentation.

- 54 mm x 85 mm Rectangle Tag (#28141)
- 50 mm Round Tag (#28142)



Actual tag dimensions may vary. Contact Parallax for specific information.

Optional Tag Information

Even though Parallax only carries a Round Tag and a Rectangle Tag the following values were obtained from different tags available in the market.

ISO Card:	6.3cm (2.5") +/- 10%
World Tag 50mm:	6.8cm (2.7") +/- 10%
World Tag 30mm:	5.3cm (2.1") +/- 10%
Bobsleigh Keyfob:	5.3cm (2.1") +/- 10%
Tear shape:	4.0cm (1.6") +/- 10%
Wristband:	4.0cm (1.6") +/- 10%

RFID Technology Overview

Material in this section is based on information provided by the RFID Journal (www.rfidjournal.com).

Radio Frequency Identification (RFID) is a generic term for non-contacting technologies that use radio waves to automatically identify people or objects. There are several methods of identification, but the most common is to store a unique serial number that identifies a person or object on a microchip that is attached to an antenna. The combined antenna and microchip are called an "RFID transponder" or "RFID tag" and work in combination with an "RFID reader" (sometimes called an "RFID interrogator").

An RFID system consists of a reader and one or more tags. The reader's antenna is used to transmit radio frequency (RF) energy. Depending on the tag type, the energy is "harvested" by the tag's antenna and used to power up the internal circuitry of the tag. The tag will then modulate the electromagnetic waves generated by the reader in order to transmit its data back to the reader. The reader receives the modulated waves and converts them into digital data. In the case of the Parallax RFID Reader Module, correctly received digital data is sent serially through the SOUT pin.

There are two major types of tag technologies. "Passive tags" are tags that do not contain their own power source or transmitter. When radio waves from the reader reach the chip's antenna, the energy is converted by the antenna into electricity that can power up the microchip in the tag (known as "parasitic power"). The tag is then able to send back any information stored on the tag by reflecting the electromagnetic waves as described above. "Active tags" have their own power source and transmitter. The power source, usually a battery, is used to run the microchip's circuitry and to broadcast a signal to a reader. Due to the fact that passive tags do not have their own transmitter and must reflect their signal to the reader, the reading distance is much shorter than with active tags. However, active tags are typically larger, more expensive, and require occasional service. The RFID Reader Module is designed specifically for low-frequency (170 kHz) passive tags.

Frequency refers to the size of the radio waves used to communicate between the RFID system components. Just as you tune your radio to different frequencies in order to hear different radio stations, RFID tags and readers have to be tuned to the same frequency in order to communicate effectively. RFID systems typically use one of the following frequency ranges: low frequency (or LF, around 170 kHz), high frequency (or HF, around 13.56 MHz), ultra-high frequency (or UHF, around 868 and 928 MHz), or microwave (around 2.45 and 5.8 GHz). It is generally safe to assume that a higher frequency equates to a faster data transfer rate and longer read ranges, but also more sensitivity to environmental factors such as liquid and metal that can interfere with radio waves.

There really is no such thing as a "typical" RFID tag. The read range of a tag ultimately depends on many factors: the frequency of RFID system operation, the power of the reader, and interference from other RF devices. Balancing a number of engineering trade-offs (antenna size v. reading distance v. power v.

manufacturing cost), the Parallax RFID Reader Module's antenna was designed with a specific inductance and "Q" factor for 170kHz RFID operation at a tag read distance of up to 1 ¾" - 3" inches.

Example Code

The following code examples read tags from a RFID Reader Module and compare the values to known tags (stored in an EEPROM table).

```
' =============================================================================
'
'   File....... RFID.BS1
'   Purpose.... RFID Tag Reader / Simple Security System
'   Author..... (c) Parallax, Inc. -- All Rights Reserved
'   E-mail..... support@parallax.com
'   Started....
'   Updated.... 07 FEB 2005
'
'   {$STAMP BS1}
'   {$PBASIC 1.0}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------
'
' Reads tags from a Parallax RFID reader and compares to known tags (stored
' in EEPROM table).  If tag is found, the program will disable a lock.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

SYMBOL  Enable          = 0                     ' low = reader on
SYMBOL  RX              = 1                     ' serial from reader
SYMBOL  Spkr            = 2                     ' speaker output
SYMBOL  Latch           = 3                     ' lock/latch control


' -----[ Constants ]-------------------------------------------------------

SYMBOL  LastTag         = 2                     ' 3 tags; 0 to 2


' -----[ Variables ]-------------------------------------------------------

SYMBOL  tag0            = B0                    ' RFID bytes buffer
SYMBOL  tag1            = B1
SYMBOL  tag2            = B2
SYMBOL  tag3            = B3
SYMBOL  tag4            = B4
SYMBOL  tag5            = B5
SYMBOL  tag6            = B6
SYMBOL  tag7            = B7
SYMBOL  tag8            = B8
SYMBOL  tag9            = B9

SYMBOL  tagNum          = B10                   ' from EEPROM table
SYMBOL  pntr            = B11                   ' pointer to char in table
SYMBOL  char            = B12                   ' character from table
```

```
' -----[ EEPROM Data ]---------------------------------------------------

Tags:
  EEPROM ("0F0184F20E")                         ' valid tags
  EEPROM ("0F01D9D263")
  EEPROM ("04129C1B43")
  EEPROM ("0000000000")                         ' space for other tags
  EEPROM ("0000000000")


' -----[ Initialization ]-------------------------------------------------

Reset:
  HIGH Enable                                   ' turn of RFID reader
  LOW Latch                                     ' lock the door!


' -----[ Program Code ]---------------------------------------------------

Main:
  LOW Enable                                    ' activate the reader
  SERIN RX, T2400, ($0A)                        ' wait for header
  SERIN RX, T2400, tag0, tag1, tag2, tag3, tag4 ' get tag bytes
  SERIN RX, T2400, tag5, tag6, tag7, tag8, tag9
  HIGH Enable                                   ' deactivate reader

Check_List:
  FOR tagNum = 0 TO LastTag                     ' scan through known tags
    pntr = tagNum * 10 + 0 : READ pntr, char    ' read char from DB
    IF char <> tag0 THEN Bad_Char               ' compare with tag data
    pntr = tagNum * 10 + 1 : READ pntr, char
    IF char <> tag1 THEN Bad_Char
    pntr = tagNum * 10 + 2 : READ pntr, char
    IF char <> tag2 THEN Bad_Char
    pntr = tagNum * 10 + 3 : READ pntr, char
    IF char <> tag3 THEN Bad_Char
    pntr = tagNum * 10 + 4 : READ pntr, char
    IF char <> tag4 THEN Bad_Char
    pntr = tagNum * 10 + 5 : READ pntr, char
    IF char <> tag5 THEN Bad_Char
    pntr = tagNum * 10 + 6 : READ pntr, char
    IF char <> tag6 THEN Bad_Char
    pntr = tagNum * 10 + 7 : READ pntr, char
    IF char <> tag7 THEN Bad_Char
    pntr = tagNum * 10 + 8 : READ pntr, char
    IF char <> tag8 THEN Bad_Char
    pntr = tagNum * 10 + 9 : READ pntr, char
    IF char <> tag9 THEN Bad_Char
    GOTO Tag_Found                              ' all match -- good tag
Bad_Char:
  NEXT

Bad_Tag:
  SOUND Spkr, (25, 80)                          ' groan
  PAUSE 1000
  GOTO Main

Tag_Found:
  DEBUG #tagNum, CR                             ' for testing
  HIGH Latch                                    ' remove latch
  SOUND Spkr, (114, 165)                        ' beep
  LOW Latch                                     ' restore latch
```



```
  GOTO Main

  END
' ------------------------------------------------------------------------
'
'   File....... RFID.BS2
'   Purpose.... RFID Tag Reader / Simple Security System
'   Author..... (c) Parallax, Inc. -- All Rights Reserved
'   E-mail..... support@parallax.com
'   Started....
'   Updated.... 07 FEB 2005
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' ------------------------------------------------------------------------


' -----[ Program Description ]---------------------------------------------
'
' Reads tags from a Parallax RFID reader and compares to known tags (stored
' in EEPROM table).  If tag is found, the program will disable a lock.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

Enable          PIN     0                       ' low = reader on
RX              PIN     1                       ' serial from reader
Spkr            PIN     2                       ' speaker output
Latch           PIN     3                       ' lock/latch control


' -----[ Constants ]------------------------------------------------------

#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T1200       CON     813
    T2400       CON     396
    T4800       CON     188
    T9600       CON     84
    T19K2       CON     32
    TMidi       CON     12
    T38K4       CON     6
  #CASE BS2SX, BS2P
    T1200       CON     2063
    T2400       CON     1021
    T4800       CON     500
    T9600       CON     240
    T19K2       CON     110
    TMidi       CON     60
    T38K4       CON     45
  #CASE BS2PX
    T1200       CON     3313
    T2400       CON     1646
    T4800       CON     813
    T9600       CON     396
    T19K2       CON     188
    TMidi       CON     108
    T38K4       CON     84
#ENDSELECT
```

```
SevenBit        CON     $2000
Inverted        CON     $4000
Open            CON     $8000
Baud            CON     T2400


#SELECT $STAMP
  #CASE BS2, BS2E
    TmAdj       CON     $100                ' x 1.0 (time adjust)
    FrAdj       CON     $100                ' x 1.0 (freq adjust)
  #CASE BS2SX
    TmAdj       CON     $280                ' x 2.5
    FrAdj       CON     $066                ' x 0.4
  #CASE BS2P
    TmAdj       CON     $3C5                ' x 3.77
    FrAdj       CON     $044                ' x 0.265
  #CASE BS2PE
    TmAdj       CON     $100                ' x 1.0
    FrAdj       CON     $0AA                ' x 0.665
  #CASE BS2Px
    TmAdj       CON     $607                ' x 6.03
    FrAdj       CON     $2A                 ' x 0.166
#ENDSELECT
```

```
LastTag         CON     3

#DEFINE   No SPRAM = ($STAMP < BS2P)        ' does module have SPRAM?


' -----[ Variables ]-------------------------------------------------------

#IF   No SPRAM #THEN
  buf           VAR     Byte(10)            ' RFID bytes buffer
#ELSE
  chkChar       VAR     Byte                ' character to test
#ENDIF

tagNum          VAR     Nib                 ' from EEPROM table
idx             VAR     Byte                ' tag byte index
char            VAR     Byte                ' character from table


' -----[ EEPROM Data ]-----------------------------------------------------

Tag1            DATA    "0F0184F20B"        ' valid tags
Tag2            DATA    "0F01D9D263"
Tag3            DATA    "04129C1B43"

Name0           DATA    "Unauthorized", CR, 0
Name1           DATA    "George Johnston", CR, 0
Name2           DATA    "Dick Miller", CR, 0
Name3           DATA    "Mary Evans", CR, 0


' -----[ Initialization ]--------------------------------------------------

Reset:
  HIGH Enable                               ' turn of RFID reader
  LOW Latch                                 ' lock the door!
```



```
' -----[ Program Code ]----------------------------------------------------

Main:
  LOW Enable                                ' activate the reader
  #IF   No SPRAM #THEN
    SERIN RX, T2400, [WAIT($0A), STR buf\10]    ' wait for hdr + ID
  #ELSE
    SERIN RX, T2400, [WAIT($0A), SPSTR 10]
  #ENDIF
  HIGH Enable                               ' deactivate reader

Check_List:
  FOR tagNum = 1 TO LastTag                 ' scan through known tags
    FOR idx = 0 TO 9                        ' scan bytes in tag
      READ (tagNum - 1 * 10 + idx), char    ' get tag data from table
      #IF __No_SPRAM #THEN
        IF (char <> buf(idx)) THEN Bad_Char     ' compare tag to table
      #ELSE
        GET idx, chkChar                    ' read char from SPRAM
        IF (char <> chkChar) THEN Bad_Char      ' compare to table
      #ENDIF
    NEXT
    GOTO Tag_Found                          ' all bytes match!

Bad_Char:                                   ' try next tag
  NEXT

Bad_Tag:
  tagNum = 0
  GOSUB Show_Name                           ' print message
  FREQOUT Spkr, 1000 */ TmAdj, 115 */ FrAdj     ' groan
  PAUSE 1000
  GOTO Main

Tag_Found:
  GOSUB Show_Name                           ' print name
  HIGH Latch                                ' remove latch
  FREQOUT Spkr, 2000 */ TmAdj, 880 */ FrAdj     ' beep
  LOW Latch                                 ' restore latch
  GOTO Main

  END


' -----[ Subroutines ]-----------------------------------------------------

' Prints name associated with RFID tag

Show_Name:
  DEBUG DEC tagNum, ": "
  LOOKUP tagNum,
         [Name0, Name1, Name2, Name3], idx      ' point to first character
  DO
    READ idx, char                          ' read character from name
    IF (char = 0) THEN EXIT                 ' if 0, we're done
    DEBUG char                              ' otherwise print it
    idx = idx + 1                           ' point to next character
  LOOP
  RETURN
```

Appendix J: Real Time Clock Chip Datasheet

DALLAS SEMICONDUCTOR

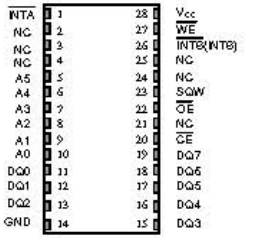
DS1286
Watchdog Timekeeper

www.dalsemi.com

FEATURES

- Keeps track of hundredths of seconds, seconds, minutes, hours, days, date of the month, months, and years; valid leap year compensation up to 2100
- Watchdog timer restarts an out-of-control processor
- Alarm function schedules real time-related activities
- Embedded lithium energy cell maintains time, watchdog, user RAM, and alarm information
- Programmable interrupts and square wave outputs maintain 28-pin JEDEC footprint
- All registers are individually addressable via the address and data bus
- Accuracy is better than ±1 minute/month at 25°C
- Greater than 10 years of timekeeping in the absence of V_{CC}
- 50 bytes of user NV RAM

PIN ASSIGNMENT



28-Pin Encapsulated Package
(720-Mil Flush)

PIN DESCRIPTION

$\overline{INTA}$	- Interrupt Output A (open drain)
$\overline{INTB}$ (INTB)	- Interrupt Output B (open drain)
A0-A5	- Address Inputs
DQ0-DQ7	- Data Input/Output
$\overline{CE}$	- Chip Enable
$\overline{OE}$	- Output Enable
$\overline{WE}$	- Write Enable
V_{CC}	- +5 Volts
GND	- Ground
NC	- No Connection
SQW	- Square Wave Output

DESCRIPTION

The DS1286 Watchdog Timekeeper is a self-contained real time clock, alarm, watchdog timer, and interval timer in a 28-pin JEDEC DIP package. The DS1286 contains an embedded lithium energy source and a quartz crystal which eliminates the need for any external circuitry. Data contained within 64 eight-bit registers can be read or written in the same manner as bytewide static RAM. Data is maintained in the Watchdog Timekeeper by intelligent control circuitry which detects the status of V_{CC} and write protects memory when V_{CC} is out of tolerance. The lithium energy source can maintain data and real time for over 10 years in the absence of V_{CC}. Watchdog Timekeeper information includes hundredths of seconds, seconds, minutes, hours, day, date, month, and year. The date at the end of the month is automatically

adjusted for months with less than 31 days, including correction for leap year. The Watchdog Timekeeper operates in either 24-hour or 12-hour format with an AM/PM indicator. The watchdog timer provides alarm windows and interval timing between 0.01 seconds and 99.99 seconds. The real time alarm provides for preset times of up to one week.

OPERATION - READ REGISTERS

The DS1286 executes a read cycle whenever $\overline{WE}$ (Write Enable) is inactive (High) and $\overline{CE}$ (Chip Enable) and $\overline{OE}$ (Output Enable) are active (Low). The unique address specified by the six address inputs (A0-A5) defines which of the 64 registers is to be accessed. Valid data will be available to the eight data output drivers within t_{ACC} (Access Time) after the last address input signal is stable, providing that $\overline{CE}$ and $\overline{OE}$ access times are also satisfied. If $\overline{OE}$ and $\overline{CE}$ access times are not satisfied, then data access must be measured from the latter occurring signal ($\overline{CE}$ or $\overline{OE}$) and the limiting parameter is either t_{CO} for $\overline{CE}$ or t_{OE} for $\overline{OE}$ rather than address access.

OPERATION - WRITE REGISTERS

The DS1286 is in the write mode whenever the $\overline{WE}$ (Write Enable) and $\overline{CE}$ (Chip Enable) signals are in the active (Low) state after the address inputs are stable. The latter occurring falling edge of $\overline{CE}$ or $\overline{WE}$ will determine the start of the write cycle. The write cycle is terminated by the earlier rising edge of $\overline{CE}$ or $\overline{WE}$. All address inputs must be kept valid throughout the write cycle. $\overline{WE}$ must return to the high state for a minimum recovery state (t_{WR}) before another cycle can be initiated. Data must be valid on the data bus with sufficient Data Set Up (t_{DS}) and Data Hold Time (t_{DH}) with respect to the earlier rising edge of $\overline{CE}$ or $\overline{WE}$. The $\overline{OE}$ control signal should be kept inactive (High) during write cycles to avoid bus contention. However, if the output bus has been enabled ($\overline{CE}$ and $\overline{OE}$ active), then $\overline{WE}$ will disable the outputs in t_{ODW} from its falling edge.

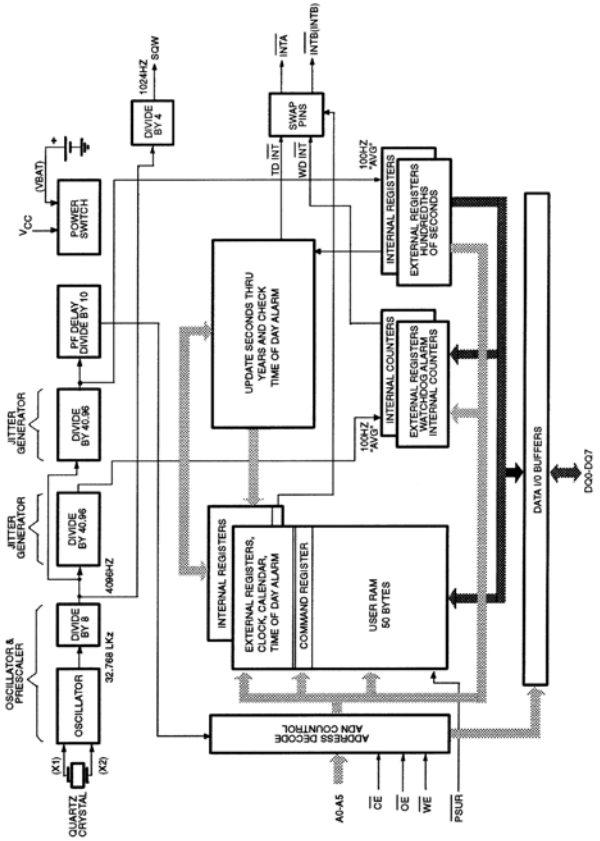
DATA RETENTION

The Watchdog Timekeeper provides full functional capability when V_{CC} is greater than 4.5 volts and write protects the register contents at 4.25 volts typical. Data is maintained in the absence of V_{CC} without any additional support circuitry. The DS1286 constantly monitors V_{CC}. Should the supply voltage decay, the Watchdog Timekeeper will automatically write protect itself and all inputs to the registers become "Don't Care." Both $\overline{INTA}$ and $\overline{INTB}$ (INTB) are open drain outputs. The two interrupts and the internal clock continue to run regardless of the level of V_{CC}. However, it is important to insure that the pull-up resistors used with the interrupt pins are never pulled up to a value which is greater than V_{CC} + 0.3V. As V_{CC} falls below approximately 3.0 volts, a power switching circuit turns on the lithium energy source to maintain the clock, and timer data functionality. It is also required to insure that during this time (battery backup mode), the voltage present at $\overline{INTA}$ and $\overline{INTB}$ (INTB) never exceeds 3.0V. At all times the current on each should not exceed +2.1 mA or -1.0 mA. However, if the active high mode is selected for $\overline{INTB}$ (INTB), this pin will only go high in the presence of V_{CC}. During power-up, when V_{CC} rises above approximately 3.0 volts, the power switching circuit connects external V_{CC} and disconnects the internal lithium energy source. Normal operation can resume after V_{CC} exceeds 4.5 volts for a period of 150 ms.

WATCHDOG TIMEKEEPER REGISTERS

The Watchdog Timekeeper has 64 registers which are 8 bits wide that contain all of the Timekeeping, Alarm, Watchdog, Control, and Data information. The Clock, Calendar, Alarm, and Watchdog registers are memory locations which contain external (user-accessible) and internal copies of the data. The external copies are independent of internal functions except that they are updated periodically by the simultaneous transfer of the incremented internal copy (see Figure 1). The Command Register bits are affected by both internal and external functions. This register will be discussed later. The 50 bytes of RAM registers can only be accessed from the external address and data bus. Registers 0, 1, 2, 4, 6, 8, 9, and A contain time of day and date information (see Figure 2). Time of Day information is stored in BCD. Registers 3, 5, and 7 contain the Time of Day Alarm information. Time of Day Alarm information is stored in BCD. Register B is the Command Register and information in this register is binary. Registers C and D are the Watchdog Alarm registers and information which is stored in these two registers is in BCD. Registers E through 3F are user bytes and can be used to contain data at the user's discretion.

BLOCK DIAGRAM Figure 1



DS1286 WATCHDOG TIMEKEEPER REGISTERS Figure 2

	ADDRESS	BIT 7						BIT 0	RANGE
CLOCK, CALENDAR, TIME OF DAY ALARM REGISTERS	0	0.1 SECONDS						0.01 SECONDS	00–99
	1	0	10 SECONDS					SECONDS	00–59
	2	0	10 MINUTES					MINUTES	00–59
	3	M	10 MIN ALARM					MIN ALARM	00–59
	4	0	12/24	10 A/P	10 HR			HOURS	01–12+A/P 00–23
	5	M	12/24	10 A/P	10 HR			HR ALARM	01–12+A/P 00–23
	6	0	0	0	0	0		DAYS	01–07
	7	M	0	0	0	0		DAY ALARM	01–07
	8	0	0	10 DATE				DATE	01–31
	9	EOSC	ESQW	0	10 MO			MONTHS	01–12
	A	10 YEARS						YEARS	00–99
COMMAND REGISTERS	B	TE	IPSW	IBH LO	PU LVL	WAM	TDM	WAF	TDF
WATCHDOG ALARM REGISTERS	C	0.1 SECONDS						0.01 SECONDS	00–99
	D	10 SECONDS						SECONDS	00–99
(RETRIGGERABLE/ REPETITIVE COUNTDOWN ALARM) USER REGISTERS	E								
	3F								

TIME OF DAY REGISTERS

Registers 0, 1, 2, 4, 6, 8, 9, and A contain Time of Day data in BCD. Ten bits within these eight registers are not used and will always read 0 regardless of how they are written. Bits 6 and 7 in the Months Register (9) are binary bits. When set to logic 0, $\overline{EOSC}$ (bit 7) enables the Real Time Clock oscillator. This bit is set to logic 1 as shipped from Dallas Semiconductor to prevent lithium energy consumption during storage and shipment. This bit will normally be turned on by the user during device initialization. However, the oscillator can be turned on and off as necessary by setting this bit to the appropriate level. Bit 6 of this same byte controls the Square Wave Output (Pin 23). When set to logic 0, the Square Wave Output pin will output a 1024 Hz Square Wave Signal. When set to logic 1 the Square Wave Output pin is in a high impedance state. Bit 6 of the Hours Register is defined as the 12- or 24- hour Select Bit. When set to logic 1, the 12-hour format is selected. In the 12-hour format, bit 5 is the AM/PM bit with logic 1 being PM. In the 24-hour mode, bit 5 is the second 10-hour bit (20-23 hours). The Time of Day registers are updated every .01 seconds from the real time clock, except when the TE bit (bit 7 of Register B) is set low or the clock oscillator is not running. The preferred method of synchronizing data access to and from the Watchdog Timekeeper is to access the Command Register by doing a write cycle to address location 0B and setting the TE bit (Transfer Enable) to a logic 0. This will freeze the External Time of Day registers at the present recorded time, allowing access to occur without danger of simultaneous update. When the watch registers have been read or written, a second write cycle to location 0B, setting the TE bit to a logic 1, will put the Time of Day registers back to being updated every 0.01 second. No time is lost in the real time clock because the internal copy of the Time of Day register buffers is continually incremented while the external memory registers are frozen.

An alternate method of reading and writing the Time of Day registers is to ignore synchronization. However, any single read may give erroneous data as the real time clock may be in the process of updating the external memory registers as data is being read. The internal copies of seconds through years are incremented and Time of Day Alarm is checked during the period that hundreds of seconds read 99 and are transferred to the external register when hundredths of seconds roll from 99 to 00. A way of making sure data is valid is to do multiple reads and compare. Writing the registers can also produce erroneous results for the same reasons. A way of making sure that the write cycle has caused proper update is to do read verifies and re-execute the write cycle if data is not correct. While the possibility of erroneous results from reads and write cycles has been stated, it is worth noting that the probability of an incorrect result is kept to a minimum due to the redundant structure of the Watchdog Timekeeper.

TIME OF DAY ALARM REGISTERS

Registers 3, 5, and 7 contain the Time of Day Alarm registers. Bits 3, 4, 5, and 6 of Register 7 will always read 0 regardless of how they are written. Bit 7 of Registers 3, 5, and 7 are mask bits (Figure 3). When all of the mask bits are logic 0, a Time of Day Alarm will only occur when Registers 2, 4, and 6 match the values stored in Registers 3, 5, and 7. An alarm will be generated every day when bit 7 of Register 7 is set to a logic 1. Similarly, an alarm is generated every hour when bit 7 of Registers 7 and 5 is set to a logic 1. When bit 7 of Registers 7, 5, and 3 is set to a logic 1, an alarm will occur every minute when Register 1 (seconds) rolls from 59 to 00.

Time of Day Alarm registers are written and read in the same format as the Time of Day registers. The Time of Day Alarm Flag and Interrupt is always cleared when Alarm registers are read or written.

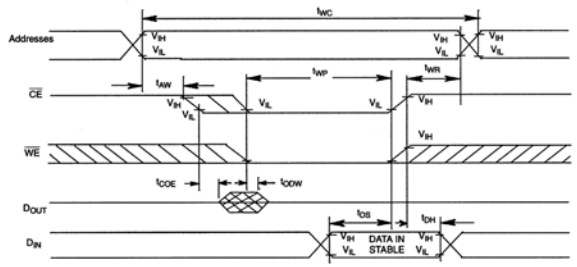
WATCHDOG ALARM REGISTERS

Registers C and D contain the time for the Watchdog Alarm. The two registers contain a time count from to 99.99 seconds in BCD. The value written into the Watchdog Alarm Registers can be written or read in any order. Any access to Registers C or D will cause the Watchdog Alarm to reinitialize and clears the Watchdog Flag bit and the Watchdog Interrupt Output. When a new value is entered or the Watchdog Registers are read, the Watchdog Timer will start counting down from the entered value to 0. When 0 is reached, the Watchdog Interrupt Output will go to the active state. The Watchdog Timer countdown is interrupted and reinitialized back to the entered value every time either of the registers is accessed. In this manner, controlled periodic accesses to the Watchdog Timer can prevent the Watchdog Alarm from ever going to an active level. If access does not occur, countdown alarm will be repetitive. The Watchdog Alarm registers always read the entered value. The actual countdown register is internal and is not readable. Writing Registers C and D to 0 will disable the Watchdog Alarm feature.

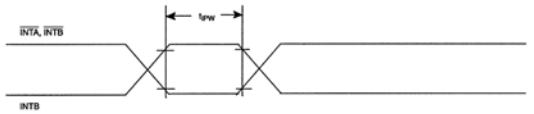
COMMAND REGISTER

Address location 0B is the Command Register where mask bits, control bits, and flag bits reside. Bit 0 is the Time of Day Alarm Flag (TDF). When this bit is set internally to a logic 1, an alarm has occurred. The time of the alarm can be determined by reading the Time of Day Alarm registers. However, if the transfer enable bit is set to logic 0 the Time of Day registers may not reflect the exact time that the alarm occurred. This bit is read only and writing this register has no effect on the bit. The bit is reset when any of the Time of Day Alarm registers are read. Bit 1 is the Watchdog Alarm Flag (WAF). When this bit is set internally to a logic 1, a Watchdog Alarm has occurred. This bit is read only and writing this register has no effect on the bit. The bit is reset when any of the Watchdog Alarm registers are accessed. Bit 2 of the Command Register contains the Time of Day Alarm Mask Bit (TDM). When this bit is written to a logic 1, the Time of Day Alarm Interrupt Output is deactivated regardless of the value of the Time of Day Alarm Flag. When TDM is set to logic 0, the Time of Day Interrupt Output will go to the active state, which is determined by bits 0, 4, 5, and 6 of the Command Register. Bit 3 of the Command Register contains the Watchdog Alarm Mask bit (WAM). When this bit is written to a logic 1, the Watchdog Interrupt Output is deactivated regardless of the value in the Watchdog Alarm registers. When WAM is set to logic 0, the Watchdog Interrupt Output will go to the active state which is determined by bits 1, 4, 5, and 6 of the Command Register. These 4 bits define how Interrupt Output Pins $\overline{INTA}$ and $\overline{INTB}$ (INTB) will be operated. Bit 4 of the Command Register determines whether both interrupts will output a pulse or level when activated. If bit 4 is set to logic 1, the pulse mode is selected and $\overline{INTA}$ will sink current for a minimum of 3 ms and then release. Output $\overline{INTB}$ (INTB) will either sink or source current for a minimum of 3 ms depending on the level of bit 5. When bit 5 is set to logic 1, the B interrupt will source current. When bit 5 is set to logic 0, the B interrupt will sink current. Bit 6 of the Command Register directs which type of interrupt will be present on interrupt pins $\overline{INTA}$ or $\overline{INTB}$ (INTB). When set to logic 1, $\overline{INTA}$ becomes the Time of Day Alarm Interrupt pin and $\overline{INTB}$ (INTB) becomes the Watchdog Interrupt pin. When bit 6 is set to logic 0, the interrupt functions are reversed such that the Time of Day Alarm will be output on $\overline{INTB}$ (INTB) and the Watchdog Interrupt will be output on $\overline{INTA}$. Caution should be exercised when dynamically setting this bit as the interrupts will be reversed even if in an active state. Bit 7 of the Command Register is for Transfer Enable (TE). The function of this bit is described in the Time of Day registers.

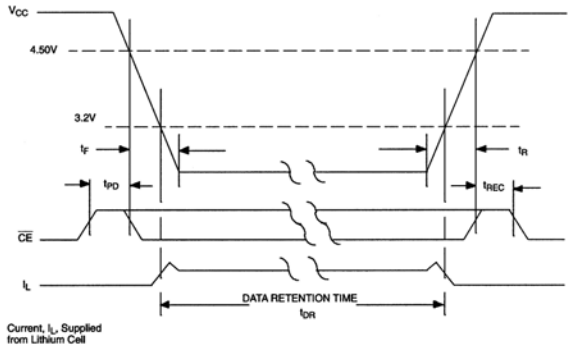
WRITE CYCLE 2 (Notes 2, 8)



TIMING DIAGRAM: INTERRUPT OUTPUTS PULSE MODE (SEE NOTES 11, 12)



POWER-DOWN/POWER-UP CONDITION



POWER-UP/POWER-DOWN CONDITION

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
$\overline{CE}$ at V_{IH} before Power-Down	t_{PD}	0			µs	
V_{CC} slew from 4.5V to 0V ($\overline{CE}$ at V_{IH})	t_F	350			µs	
V_{CC} slew from 0V to 4.5V ($\overline{CE}$ at V_{IH})	t_R	100			µs	
$\overline{CE}$ at V_{IH} after Power Up	t_{REC}			150	ns	

(t_A=25°C)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Expected Data Retention Time	t_{DR}	10			years	9

WARNING:

Under no circumstances are negative undershoots, of any amplitude, allowed when device is in battery backup mode.

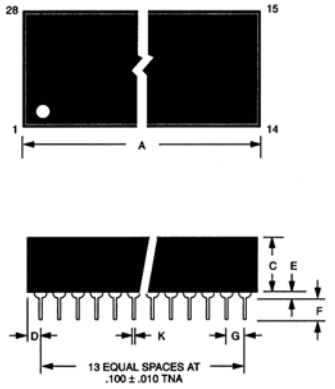
NOTES:

1. $\overline{WE}$ is high for a read cycle.
2. $\overline{OE}$ = V_{IH} or V_{IL}. If $\overline{OE}$ = V IH during write cycle, the output buffers remain in a high impedance state.
3. t_{WP} is specified as the logical AND of the $\overline{CE}$ and $\overline{WE}$. t_{WP} is measured from the latter of $\overline{CE}$ or $\overline{WE}$ going low to the earlier of $\overline{CE}$ or $\overline{WE}$ going high.
4. t_{DS} or t_{DH} are measured from the earlier of $\overline{CE}$ or $\overline{WE}$ going high.
5. t_{DH} is measured from $\overline{WE}$ going high. If $\overline{CE}$ is used to terminate the write cycle, then t_{DH} = 20 ns.
6. If the $\overline{CE}$ low transition occurs simultaneously with or later than the $\overline{WE}$ low transition in Write Cycle 1, the output buffers remain in a high impedance state during this period.
7. If the $\overline{CE}$ high transition occurs prior to or simultaneously with the $\overline{WE}$ high transition, the output buffers remain in a high impedance state during this period.
8. If $\overline{WE}$ is low or the $\overline{WE}$ low transition occurs prior to or simultaneously with the $\overline{CE}$ low transition, the output buffers remain in a high impedance state during this period.
9. Each DS1286 is marked with a four-digit date code AABB. AA designates the year of manufacture. BB designates the week of manufacture. The expected t_{DR} is defined as starting at the date of manufacture.
10. All voltages are referenced to ground.
11. Applies to both interrupt pins when the alarms are set to pulse.
12. Interrupt output occurs within 100 ns on the alarm condition existing.
13. Both $\overline{INTA}$ and $\overline{INTB}$ (INTB) are open drain outputs.
14. Real-Time Clock Modules can be successfully processed through conventional wave-soldering techniques as long as temperature exposure to the lithium energy source contained within does not exceed +85°C. Post-solder cleaning with water washing techniques is acceptable, provided that ultrasonic vibration is not used.

AC TEST CONDITIONS

Output Load: 100 pF + 1TTL Gate
Input Pulse Levels: 0-3.0V
Timing Measurement Reference Levels
Input: 1.5V
Output: 1.5V
Input Pulse Rise and Fall Times: 5 ns.

DS1286 WATCHDOG TIMEKEEPER



PKG	28-PIN	
DIM	MIN	MAX
A IN.	1.520	1.540
MM	38.61	39.12
B IN.	0.695	0.720
MM	17.65	18.29
C IN.	0.350	0.375
MM	8.89	9.52`
D IN.	0.100	0.130
MM	2.54	3.30
E IN.	0.015	0.030
MM	0.38	0.76
F IN.	0.110	0.140
MM	2.79	3.56
G IN.	0.090	0.110
MM	2.29	2.79
H IN.	0.590	0.630
MM	14.99	16.00
J IN.	0.008	0.012
MM	0.20	0.30
K IN.	0.015	0.021
MM	0.38	0.53

NOTE: PINS 2,3,21,24 AND 25 ARE MISSING BY DESIGN

Appendix K: Motor Datasheet

4118

- NEMA Size 17 Mounting
- Cost Effective
- Custom Windings Available (No Additional Cost)

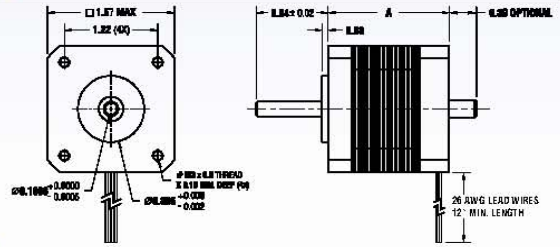
SPECIFICATIONS

	Dimension "A" Max	Model #	Amps/ Phase	Torque oz-in	Torque N-m	Resistance Ohm/Phase	Inductance mH/Phase	Inertia oz-in²	Weight Lbs.	Number of Leads
BIPOLAR	1.35" 34.3 mm	4118S-02	1.30	45.0	0.32	2.8	3.6	0.18	0.40	4
		4118S-04S	0.67	45.0	0.32	9.9	12.5	0.18	0.40	4
		4118S-04P	1.33	45.0	0.32	2.5	3.1	0.18	0.40	4
		4118S-09	0.90	45.0	0.28	5.3	6.7	0.18	0.40	4
	1.57" 39.9 mm	4118M-01	1.70	63.0	0.44	1.5	3.0	0.28	0.60	4
		4118M-06S	0.70	63.0	0.44	10.8	21.8	0.28	0.60	4
		4118M-06P	1.40	63.0	0.44	2.7	5.5	0.28	0.60	4
	1.9" 48.3 mm	4118L-01	2.00	83.0	0.59	1.4	2.7	0.37	0.70	4
		4118L-07S	1.05	83.0	0.59	5.2	9.4	0.37	0.70	4
		4118L-07P	2.10	83.0	0.59	1.3	2.3	0.37	0.70	4

	Dimension "A" Max	Model #	Amps/ Phase	Torque oz-in	Torque N-m	Resistance Ohm/Phase	Inductance mH/Phase	Inertia oz-in²	Weight Lbs.	Number of Leads
UNIPOLAR	1.35" 34.3 mm	4118S-04	0.95	30.0	0.17	5.0	3.1	0.18	0.40	6
	1.57" 39.9 mm	4118M-06	1.00	45.0	0.35	5.4	5.5	0.28	0.60	6
	1.9" 48.3 mm	4118L-07	1.50	65.0	0.43	2.6	2.3	0.37	0.70	6
		4118L-25	0.45	65.0	0.35	25.0	17.4	0.37	0.70	6

- Please complete our application data sheet for different windings.
- Power supply voltage can be any value as long as the driver output current is controlled at the rated current.
- Call Lin Engineering for additional bipolar torque curves.
- Performance, size, and appearance specifications of the products listed here are subject to change without notice.
- For operating temperatures, see page 94.

DIMENSIONS (inches)

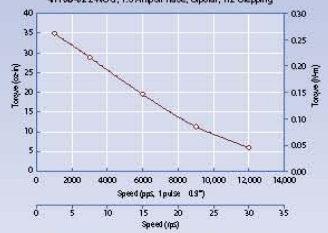


DID YOU KNOW...

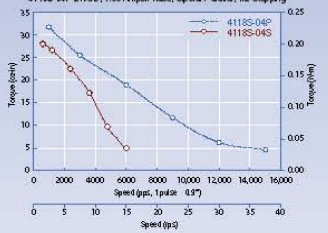
The average lead time for prototypes is 3-7 business days. This includes custom windings and most shaft modifications.

TORQUE CURVES

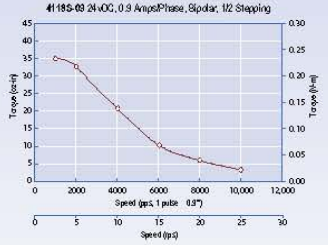
4118S-02 24vDC, 1.3 Amps/Phase, Bipolar, 1/2 Stepping



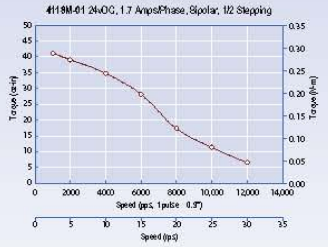
4118S-04S 24vDC, 0.67 Amps/Phase, Bipolar Series, 1/2 Stepping
4118S-04P 24vDC, 1.33 Amps/Phase, Bipolar Parallel, 1/2 Stepping



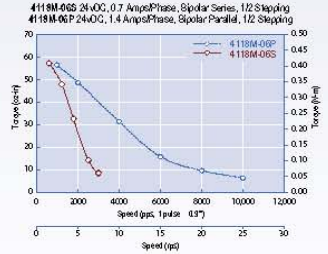
4118S-09 24vDC, 0.9 Amps/Phase, Bipolar, 1/2 Stepping



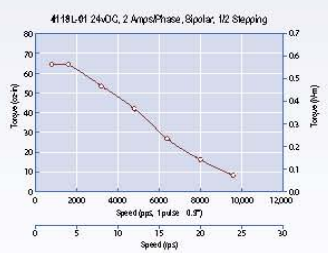
4118M-01 24vDC, 1.7 Amps/Phase, Bipolar, 1/2 Stepping



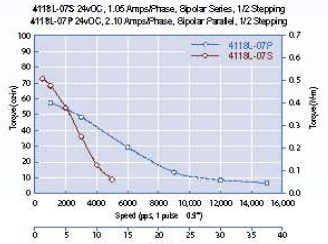
4118M-06S 24vDC, 0.7 Amps/Phase, Bipolar Series, 1/2 Stepping
4118M-06P 24vDC, 1.4 Amps/Phase, Bipolar Parallel, 1/2 Stepping



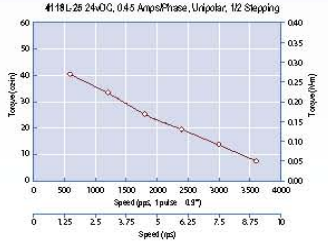
4118L-01 24vDC, 2 Amps/Phase, Bipolar, 1/2 Stepping



4118L-07S 24vDC, 1.05 Amps/Phase, Bipolar Series, 1/2 Stepping
4118L-07P 24vDC, 2.10 Amps/Phase, Bipolar Parallel, 1/2 Stepping



4118L-25 24vDC, 0.45 Amps/Phase, Unipolar, 1/2 Stepping



Appendix L: 24V Power Supply Datasheet

Input Specifications

Input Voltage Range	85-264 VAC
Input Current :	3.15 A/ 115 VAC 1.5A /230V
Frequency Range:	Range:47-63 Hz
Inrush Current, typ:	Cold Start 30A/115 VAC
Leakage Current	<=1mA/ 240 VAC

Output Specifications

Voltage and Current	See Selection Chart
Load Reulation (0%-FL)	+/- 0.5 %
Line Regulation	+/- 0.5%
Voltage Tolerance	+/- 1-0%
Voltage Adjustment Range	2.16 ~ 26.4V
Ripple/ Noise (max)	150mVp-p
Over Voltage Protection	Shutdown; Re-power on
Overload Protection	105 ~ 150%, Constant 1 Limit Auto Recovery
Setup/ Rise/ Holdup (230 VAC)	1s, 30ms at full load

General Specifications

Input-Out Isolation	I/P-O/P: 300 VAC I/P-G: 1500 VAC O/P: 0.5KVAC
Isolation Resistance I/P-O/P, I/P-G O/P-G:	500VDC/100 Ohm
Efficiency (3.3 V through 48V)	83% typ
Switching Frenquency	135 Khz, (fixed, typical)
Safety	
EN60950	TUV File#9758434
UL1950	UL# E183223

Environmental Specifications

Oper. Temperature	-10 ~ +60°C
Storage Temperature	-20 to +85°C
Relative Humidity	10% to 95%, non-cond *
Vibration	10 ~ 500Hz, 2G 10min./ 1cycle, Period for 60 min. Each Axes
EMC	CISPR22 (EN55022)B, EN61000-4-2,3,5,6,8,11 ENV50204 EN61000-3-2,-3
MTBF	314,900 Hrs

Physical Specifications

Size	3.0" x 8.7" x 1.6"
Contruction	Closed Frame
Weight	Weight 19.4 oz, (550g)